JPRS 82535

27 December 1982

USSR Report

CYBERNETICS, COMPUTERS AND AUTOMATION TECHNOLOGY

No. 66

FBIS FOREIGN BROADCAST INFORMATION SERVICE

NOTE

JPRS publications contain information primarily from foreign newspapers, periodicals and books, but also from news agency transmissions and broadcasts. Materials from foreign-language sources are translated; those from English-language sources are transcribed or reprinted, with the original phrasing and other characteristics retained.

Headlines, editorial reports, and material enclosed in brackets [] are supplied by JPRS. Processing indicators such as [Text] or [Excerpt] in the first line of each item, or following the last line of a brief, indicate how the original information was processed. Where no processing indicator is given, the information was summarized or extracted.

Unfamiliar names rendered phonetically or transliterated are enclosed in parentheses. Words or names preceded by a question mark and enclosed in parentheses were not clear in the original but have been supplied as appropriate in context. Other unattributed parenthetical notes within the body of an item originate with the source. Times within items are as given by source.

The contents of this publication in no way represent the policies, views or attitudes of the U.S. Government.

PROCUREMENT OF PUBLICATIONS

JPRS publications may be ordered from the National Technical Information Service (NTIS), Springfield, Virginia 22161. In ordering, it is recommended that the JPRS number, title, date and author, if applicable, of publication be cited.

Current JPRS publications are announced in Government Reports Announcements issued semimonthly by the NTIS, and are listed in the Monthly Catalog of U.S. Government Publications issued by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

Correspondence pertaining to matters other than procurement may be addressed to Joint Publications Research Service, 1000 North Glebe Road, Arlington, Virginia 22201.

JPRS 82535

27 December 1982

USSR REPORT

CYBERNETICS, COMPUTERS AND AUTOMATION TECHNOLOGY

No. 66

CONTENTS

[III - USSR - 21C S&T]

APPLICATIONS

ORGANIZATIONS

CONFERENCES

UDC 621.3.049.77:621.9.048.7

ELECTRON BEAM LITHOGRAPHY UNIT CONTROL SYSTEM

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 (manuscript received 28 Jan 82) pp 56-58

[Article by Vyacheslav Nikolayevich V'yukhin, senior scientific associate, IAE SO AN SSSR [Institute of Atomic Energy imeni I.V. Kurchatov, Siberian Division, USSR Academy of Sciences], Novosibirsk; Aleksandr Nikolayevich Kasperovich, engineer, IAE SO AN SSSR, Novosibirsk; Yevgeniy Aleksandrovich Kovalev, engineer, IAE SO AN SSSR, Novosibirsk; Vladimir Ivanovich Prokopenko, engineer, IAE SO AN SSSR, Novosibirsk; and Vladimir Pavlovich Yunoshev, engineer, IAE SO AN SSSR, Novosibirsk]

[Text] One of the most important aims in the problem of introducing electron beam lithography into practice is improvement of the productivity of electron beam lithography (ELL) units. With electron beam current densities of 1 to 10 A/cm^2 and electron resist sensitivity of 10^{-6} to 10^{-8}C per cm^2 the required exposure time for a single point equals 10^{-6} to 10^{-9} s and consequently the unit's productivity will be determined practically completely by the control system. The control system must enable rapid movement of the beam to a specified position with a maximum electronic scanning field and enable the construction of elementary patterns without participation of an external computer.

The "world standard" in the area of controlling ELL vector units is the use of high-speed 16-bit digital-analog converters, the development of which represents a fairly complicated engineering problem.

The control system (fig 1) is designed according to the CAMAC standard and contains the following modules: two 16-bit digital-analog converters; two deflection current amplifiers; one beam current or exposure dose meter; one special controller; one video amplifier making the REM [scanning electron microscope] mode possible in conjunction with a monitor.

All modules are placed in a CAMAC crate which is connected to the control computer via a crate controller. The individual modules of the system and their characteristics will be described below.

The TsAP-16 [digital-analog converter] has been specially developed for controlling the position of the electron beam in high-resolution electron beam units. It is described in detail in [1] and has the following characteristics:

Output signal	
In unipolar mode	0 to 8.192 V
In bipolar mode	$\pm$ 4.096 V
Quantization level	125 μV
Switching time	
With low signal	3 μs
With high signal	10 μs
Differential and integral nonlinearity	0.001 percent

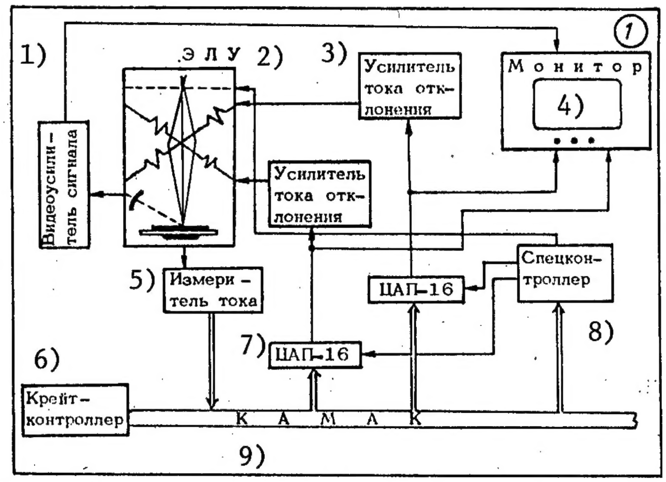


Figure 1. Block Diagram of Electron Beam Lithography Unit Control System

Key:

1. Signal video amplifier
2. Electron beam lithography unit
3. Deflection current amplifier
4. Monitor
5. Current meter
6. Crate controller
7. TsAP-16 [digital-analog converter]
8. Special controller
9. CAMAC

For the purpose of correcting scale distortions of the electron beam deflection section and referencing to datum marks, a mode is provided in the TsAP-16 for electronic adjustment of the conversion scale within limits from the nominal while preserving metrological properties. The TsAP-16 is placed in a module with a width of 2M and executes instructions of the REGISTER trunk, +1 and -1.

The deflection current amplifier is designed according to a circuit with a resistor current-measuring element in a feedback loop. The current amplifier (fig 2) contains in its structure a low-power wideband d.c. amplifier, current amplification power stages, feedback resistors R_1 to R_3, and a modulation-demodulation (MDM) channel which stabilizes zero drift.

The d.c. amplifier is executed with discrete components since one commercial monolithic operational amplifier does not make possible the required speed of response. Feedback resistors R_1 and R_2 are matched film resistors, and resistor R_3--the current-measuring element--is formed from 20 1-W S5-5 resistors placed in a special container--a heat sink.

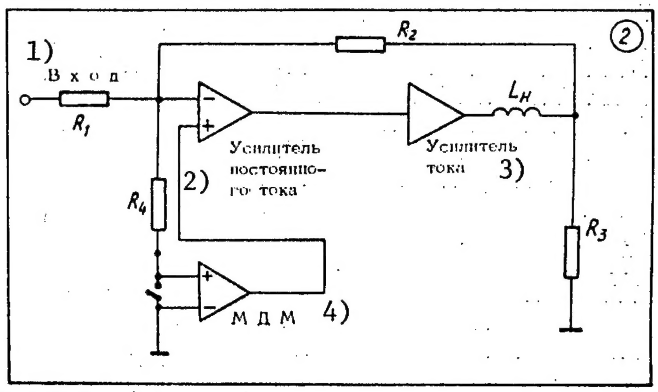


Figure 2. Circuit Diagram of Current Amplifier

Key:

1. Input
2. d.c. amplifier
3. Current amplifier
4. MDM

When the current amplifier operates in the high-signal mode, impulse noise originates in its inverting input, caused by delay of the negative feedback signal. This noise, penetrating the MDM channel, causes dynamic zero drift proportional to the frequency and amplitude of the input signal. In the circuit diagram in fig 2 the switch, K, short-circuits the inputs of the MDM channel for the time of the transient process in the main channel, eliminating the transmission of impulse noise in the MDM channel.

When operating on a single-cycle deflection system with inductance of 0.5 mH the characteristics of the current amplifier should be as follows: current-- $\pm$ 0.8 A; linearity and stability--better than 0.005 percent; switching time--3 μs for low signal and 40 μs for full signal.

The beam current (exposure dose) meter enables uniform distribution of the charge over all fragments of the pattern regardless of the configuration of fragments. For the purpose of implementing these conditions, a number of charge-measuring modules are used in the system, making possible a programmed quantitative estimate of the value of the charge added to the fragment under consideration.

Simple estimates have demonstrated that the sensitivity of the converter to a charge should be not worse than 10^{-11} C with a signal dynamic range on the order of 10^6.

As a charge meter the best idea is to use a current-frequency converter (PTCh) in combination with a counter and control unit.

The structural diagram of the current-frequency converter is shown in fig 3. Here the input current signal is integrated by means of capacitance C_n. In a particular case this capacitance can be the capacitance of the structural elements of the microscope stage and the coaxial connecting cable. Integration is performed to the threshold level, U_{por}, assigned by the comparator. When the threshold level

is reached, a device for forming a compensating charge utilizing capacitor C_q and two diodes, D_1 and D_2, produces a reference charge, q_0, which enters the integrating capacitor and compensates the measured charge. Charge q_0 represents the unit reference standard in the current-frequency converter.

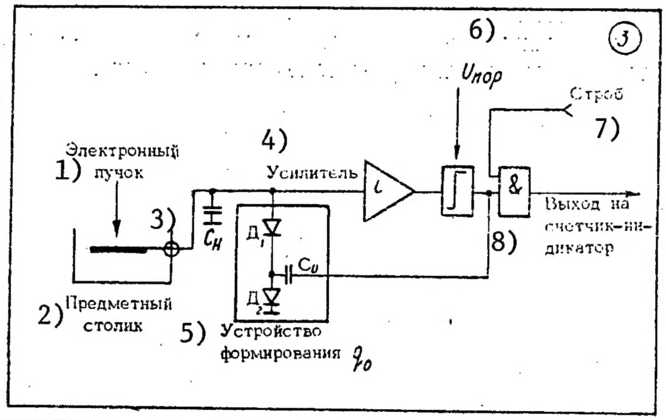


Figure 3. Circuit Diagram of Beam Current (Exposure Dose) Meter

Key:

1. Electron beam
2. Microscope stage
3. C_n
4. Amplifier
5. Forming unit
6. U_{por} [threshold]
7. Gate
8. Output to counter-display

A gate pulse is fed to the input of the AND circuit, permitting the output train of current-frequency converter pulses to enter the input of the counter. This gate pulse is synchronous and is equal in duration to the beam blanking signal. The digital equivalent of the current charge is formed by the counter and a reference interval former.

The technical characteristics of the beam current or exposure dose meter are as follows:

Sensitivity to charge	10^{-11} C
Conversion transconductance	10^{11} Hz/A
Dynamic range of input signals	$0.5 \cdot 10^{6}$
Fundamental error	Not greater than 1 percent

The special controller relieves the computer of performing operations of forming elementary patterns. It makes it possible to construct elementary fragments (vectors, arcs and rectangles) and to form the beam blanking signal. Out of considerations of speed and word length the special controller is executed on the basis of a K589 microprocessor and is microprogrammable.

The structural arrangement of the special controller consists of a microprocessor section (microprogam control unit, eight 2-bit central processors with a fast carry circuit, a microinstruction ROM and a condition multiplexer) and an interface section (a CAMAC interface matching the format for the representation of instructions and data in the CAMAC trunk with the format used in the microprocessor, as well as

a digital-analog converter interface through which control of the beam's position and formation of the blanking pulse take place).

The estimate function method is used in the controller in constructing vectors and arcs. On the analytical line estimate function U equals zero; with steps along axes X and Y the estimate function becomes different from zero. By analyzing the sign of the function it is possible to determine along which axis the next step must be made.

In the vector construction mode the estimate function is computed in the following manner with steps along axes X and Y, respectively:

$$X_{i+1} = X_i + 1; \quad Y_{i+1,j} = U_{i,j} - \Delta Y;$$
$$Y_{i+1} = Y_j + 1; \quad U_{i,j+1} = U_{i,j} + \Delta X,$$

where $U_{i,j}$ is the value of the estimate function at a point with coordinates (X_i, Y_i); ΔY is the projection of the vector which is to be constructed onto the Y axis; and ΔX is the projection of the vector onto the X axis.

In the arc construction mode the estimate function is determined with steps along axes X and Y, respectively, as follows:

$$X_{i+1} = X_i + 1; \quad U_{i+1,j} = U_{i,j} + 2X_i + 1;$$
$$Y_{i+1} = Y_j + 1; \quad U_{i,j+1} = U_{i,j} - 2Y_j + 1.$$

In all cases the initial value of the estimate function is assumed to equal zero. The initial coordinates of the fragment are transferred directly to the TsAP-16. In the vector construction mode the values of ΔX and ΔY are transferred to the controller and in the arc construction mode the coordinates of the starting point, X_n, Y_n $(X_n^2 + Y_n^2 = R^2)$, and the number of steps along X and Y are entered, which equals three CAMAC instructions. Direct construction of fragments is accomplished by feeding from the controller to the TsAP-16 unit increments.

The computing cycle for a single point consists of five microinstructions, four of which are associated with computation of trajectory parameters and one is the check operation. The presence of several internal registers in the central processor makes it possible to store initial parameters, which makes it possible to manage with only one start instruction in constructing subsequent identical patterns. After computation of the next point and movement of the beam along the appropriate coordinate, a pause is formed by hardware means for the period of the transient process in the analog section of the system and then a blanking pulse of programmable duration is formed. During the pause and blanking period the processor computes the next point and if the blanking signal has still not ended, goes into the waiting state.

Construction of rectangles is based on an algorithm for repeated reproduction of the vector along one of the coordinates with a simultaneous step-by-step movement along the other coordinate.

In the controller provision is made for cyclic reproduction of an elementary pattern which, firstly, is useful in checking the working ability of the controller (e.g., with an oscillograph) and, secondly, for enabling the scanning mode of operation of the electron beam unit as an electron microscope.

The controller has the following characteristics:

Number format	16 bits
Time for execution of microinstructions	280 ns
Maximum time for computing a point	1.4 μs
Range of programmable duration of blanking signal	4.5 μs to 1.3 ms
Maximum size	
Of vector	2^{15} steps
Of arc	2^{14} steps
Of rectangle	2^{16} X 2^{16} steps
Error in computing coordinates of points of any pattern	Not greater than 1 step
Capacity of ROM occupied by microprograms for implemented patterns	126 locations
Capacity of ROM for possible additional algorithms (e.g., tag search)	130 locations

The special controller is designed according to the CAMAC standard with a width of 4M and is controlled by a set of 10 instructions.

The use of the special controller makes it possible to increase by approximately an order of magnitude the maximum speed of the beam control system as compared with the variant in which an SM-3 computer is the controller. Simple operations (e.g., the scanning mode mentioned) can be performed in the off-line mode.

The control system developed was used as part of the JBX-2 unit.

8831
CSO: 1863/1

UDC 537.312.6:621.382

HIGH-VOLTAGE ELECTRON BEAM LITHOGRAPHY USING WAFERS AND THIN MEMBRANES

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 (manuscript received 28 Jan 82) pp 58-59

[Article by Grigoriy Trofimovich Sbezhnev, candidate of technical sciences, Moscow, and Nikolay Fedorovich Pesotskiy, engineer, Moscow]

[Text] According to the results of numerous theoretical and experimental studies the minimum width of lines which can be produced by electron beam lithography with the usual accelerating voltages of 10 to 30 kV is restricted by the scattering of electrons [1]. Lines 0.2 to 0.3 μ wide have been produced in practice by optimizing exposure and development conditions in a resist 0.5 to 1 μ thick, but these results are insufficiently reproducible and have been poorly tested.

Recently a tendency has been noted toward studying the electron beam lithography process with higher accelerating voltage values. For example, in [2] it was demonstrated that with an accelerating voltage of 50 kV the reproducibility of results is improved and the profile of lines has an almost rectangular shape in rather thick (about 1 μ) layers of the resist. It is important that with an increase in accelerating voltage the speed of applying the pattern is not reduced, since the reduction in sensitivity of the resist is compensated by an increase in current density in the electron beam [3]. Even higher quality of the pattern is achieved when using substrates in the form of a thin membrane [4], which transmits electrons in beams practically without reflection.

We have put together a mockup of a vector scanning unit with accelerating voltage of 40, 60, 80 and 100 kV and a minimum beam diameter of 0.05 μ. Development and study of the process of high-voltage electron beam lithography are under way at the present time on the basis of this unit.

The purpose of the studies being made is to achieve sufficiently high quality of the pattern and reproducibility with resolution at the 0.1 μ level using resist thicknesses which are technologically acceptable at the present time.

An EP-1 positive resist (based on polymethyl methacrylate) 0.35 to 0.4 μ thick, which was applied to silicon wafers and polyimide membranes 1 to 2 μ thick with a metal precoat, was exposed. After exposure the samples were developed in a mixture of isopropyl alcohol and methyl ethyl ketone in a 3:1 ratio. The temperature of the developer was kept within the range of 22 ± 1 °C and the development time was 45 ± 5 s.

The minimum width of developed lines on wafers and membranes was achieved with accelerating voltages of 80 to 100 kV and equaled 0.08 μ. The results obtained with membranes were distinguished by very good reproducibility. With an increase in the exposure dose by 30 percent over the nominal (10^{-8} C/cm with U = 80 kV) line broadening practically was not observed. For wafers the permissible change in dose is somewhat lower--15 to 20 percent. It must mentioned, however, that the accuracy of measuring the width of lines on a scanning electron microscope was approximately 0.01 μ.

The influence of the reflected electrons on the nature of development of the pattern was observed with accelerating voltages right up to 100 kV. In series of closely placed lines this influence was evidenced in line broadening and in a reduction in thickness of the developed resist toward the center of the series.

The effect of proximity is evidenced differently in development of a pattern depending on the accelerating direction [as published]. With low values of U (20 kV) distortion of the geometrical boundaries of elements is observed in topologies with high density of elements. These distortions are reduced with an increase in accelerating voltage and are practically absent at 100 kV, but the thickness of the developed resist is reduced by approximately 30 percent almost uniformly over the entire field of the pattern.

This is explained by the fact that with an increase in U the distance, R , at which the reflected electrons act is increased approximately proportional to $U^{1.75}$ and with 100 kV R $\underset{-}{\sim}$ 30 μ , which is comparable with and even exceeds the typical interleaving intervals of elements of the topology of large-scale integrated circuits with a high degree of integration. As a result, averaging of the density of the energy scattered by reflected electrons in the resist over the field of the pattern takes place. And the total energy surrendered to the resist by reflected electrons, as calculations have demonstrated, almost does not depend on the accelerating voltage in the range from 20 to 100 kV.

With membranes the proximity effect is practically completely absent with 60 kV and higher, which made it possible to produce without distortion spaces between elements 0.1 μ wide. With lower values of the width of a space strips of the resist not dissolved in the developer were washed away because of insufficient adhesion of the resist to the membrane.

At the present time the process of high-voltage electron beam lithography using membranes is being used to develop a technology for producing x-ray masks. Experimental samples of x-ray masks with a minimum line width of 0.08 to 0.09 μ with a gold mask thickness of 0.12 μ have been produced by the detonation method. This thickness of the gold is sufficient, for example, when using carbon K-line x-radiation ($\lambda \underset{-}{\sim}$ 4.5 nm).

The products produced can be used as electron masks for a projection-type electron beam unit operating with transmission, which is being developed at the present time. In this unit the formation of contrast takes place similarly to how it does in a transmission electron microscope, but the contrast achieved is considerably greater than in x-ray lithography with a gold thickness of not less than 0.1 μ.

The application of high-voltage electron beam lithography using membranes is not restricted to making x-ray and electron masks. There are certain devices for whose implementation dimensions of less than 0.1 μ are required, which can be achieved with good reproducibility by means of high-voltage electron beam lithography using membranes made of silicon or other materials.

For products utilizing thin membranes an effective testing method exists using a transmission scanning microscope which has higher resolution and image contrast as compared with an ordinary scanning microscope. Use of the domestic PREM-200 instrument for this purpose, which has resolution at the level of 30 Å, will make it possible to improve further the promising technological process of high-voltage electron beam lithography utilizing thin membranes.

Bibliography

1. Murrae, J.B. "Electron Irradiation of Polymer and Its Application to Resists for Electron Beam Lithography," CRITICAL REV. IN SOLID STATE AND MATTER SC., No 3, 1978, pp 223-264.

2. Neill, T.R. and Bull, C.J. "High Voltage Electron Lithography," ELECTR. LETT., Vol 16, No 16, 1980, pp 621-623.

3. Bokov, Yu.S. and Sbezhnev, G.T. "Sensitometry of Electron Resists," ELEKTRONNAYA TEKHNIKA, Ser. 3, No 6, 1980, pp 74-78.

4. Broers, A.N. "Resolution Limit of PMMA Resist for Exposure with 50 kV Electrons," J. ELECTROCHEM. SOC.: SOLID-STATE SC. AND TECHN., Vol 128, No 1, 1981, pp 166-170.

8831
CSO: 1863/1

UDC 537.212.6:621.382

COMBINATION OF OPTICAL AND ELECTRON BEAM LITHOGRAPHY IN FABRICATION OF BRIDGE-TYPE JOSEPHSON ELEMENTS

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 (manuscript received 28 Jan 82) pp 60-61

[Article by Aleksey Aleksandrovich Altukhov, candidate of physical and mathematical sciences, FIAN SSSR [Physics Institute imeni P.N. Lebedev, USSR Academy of Sciences], Moscow; Yuriy Petrovich Baryshev, engineer, FIAN SSSR, Moscow; Kamil' Akhemtovich Valiyev, corresponding member, USSR Academy of Sciences, FIAN SSSR, Moscow; Leonid Vasil'yevich Velikov, candidate of physical and mathematical sciences, FIAN SSSR, Moscow; Rim Khakimovich Makhmutov, engineer, FIAN SSSR, Moscow; Aleksandr Aleksandrovich Orlikovskiy, candidate of technical sciences, FIAN SSSR, Moscow; Irina Sergeyevna Sokolova, engineer, FIAN SSSR, Moscow; and Nikolay Yur'yevich Ustinov, engineer, FIAN SSSR, Moscow]

[Text] Introduction

One promising scientific trend in microelectronics is the creation and study of devices based on the Josephson effect for digital and measuring equipment. Superconducting quantum interferometers [1] making it possible to measure magnetic fields (to 10^{-10} Gs) or low voltage (to 10^{-15} V) have received practical application. Logic and memory large-scale integrated circuits employing Josephson tunnel junctions have also been described, whose speed of response for the time being is essentially not greater than that of semiconductor large-scale integrated circuits.

An analysis of the data in [2, 3] has demonstrated that the data which have been accumulated in research on digital Josephson elements can serve only as a starting point for further research aimed at creation of improved digital superlarge-scale integrated circuits utilizing the Josephson effect.

Remaining a complicated technological problem is the creation of stable Josephson junctions utilizing electron beam lithography, ion implantation and dry etching [4], making possible satisfactory reproducibility of the parameters of junctions.

Creation of high-quality bridge-type Josephson junctions is practically impossible without using electron beam lithography (ELL). The use of ELL for creating bridge junctions in films of Al, Sn and Pb was described for the first time in [5]. However, the use of ELL for exposing an entire Josephson device is not feasible because of the low productivity of the process. In our opinion combining the ELL and photolithography processes is promising. In this case photolithography is used for

creating superconducting "banks" with linear dimensions of about 10 μ and bonding pads; a "slight" coupling section--a bridge with submicron dimensions--is created by means of ELL.

In this study Sn is used as the material for a bridge-type junction. Of course, Sn films are less resistant to thermal cycling than Nb films are. However, Sn has higher values of the order parameter, ξ , which makes it possible to simplify the route for creating a junction, e.g., as compared with those described in [4, 6].

Sketch of Process of Creating Bridge Junction Based on Tin

A diagram is presented in fig 1 for the technological process for a bridge-type Josephson junction utilizing Sn and employing ELL and plasma chemical etching.

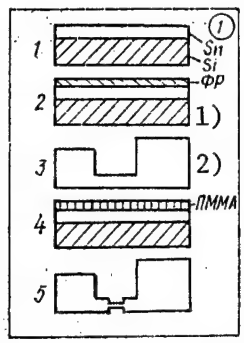


Figure 1. Process of Creating Bridge Junction: 1--vacuum thermal spraying; 2--photolithography; 3--plasma chemical etching; 4--electron beam exposure; 5--plasma chemical etching

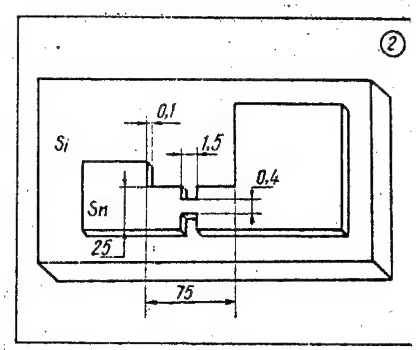


Figure 2. Josephson Bridge Junction Based on Film (Dimensions Given in μ)

Key:

1. Photoresist
2. PMMA [polymethyl methacrylate]

A film of Sn 0.1 μ thick is applied by the vacuum thermal method to a silicon substrate. By means of standard optical lithography and plasma chemical etching a structure is created consisting of two bonding pads connected by a bridge 73 μ long and 25 μ wide. By means of ELL a submicron gap (fig 2) is created in the bridge. The width of the exposure half-slit is 1.5 μ and the distance between half-slits is 0.4 μ.

Application of Tin Film

By the method of thermal spraying a film of Sn was applied under vacuum from a tantalum crucible. The substrate was a wafer of monocrystalline silicon with (111) orientation 50 mm in diameter. The working vacuum was $8 \cdot 10^{-6}$ torr. The distance from the vaporizer to the substrate was 350 mm. With substrate temperatures from

room to 80 °C (measuring accuracy ± 5 °C) fine-grain films were produced with a characteristic metallic gloss and above 80 °C the films were coarse-grained and mat.

The rate of deposition of the film was 2.5 nm/s. The presence of silicides of tin was not detected in the film by Auger spectrum analysis (up to 110 °C). The temperature for transition of the film into the superconducting state equaled 3.72 °K.

Plasma Chemical Etching

Modern submicron technology assumes the use of dry etching processes such as ionic, ionic-chemical and plasma chemical. The specific type of dry etching is selected as a function of the material, the required profile and the dimensions of the structure. Tin is a low-melting metal (232 °C) which creates difficulties in using ionic and ionic-chemical etching, since the temperature of the substrate in the process of physical spraying of materials is increased to hundreds of degrees.

Experiments using ionic and ionic-chemical etching units have shown also that under the influence of high-energy particles irreversible changes take place in the tin resist structure. The etching process is either stopped completely or degradation of the resistive mass takes place.

The use of plasma chemical etching is optimal, which was carried out in a cavity reaction vessel with a capacitor by means of high-frequency excitation of the plasma. The frequency of the high-frequency oscillator was 13.56 MHz and the power input into the charge was 1 kW. Sn films were etched in a C Cl gas-discharge plasma. The pressure of residual gases equaled 10^{-1} torr. The C Cl partial vapor pressure was $5 \cdot 10^{-2}$ torr. The average etching rate for the tin was 1.5 nm/s.

Electron Beam Exposure

After plasma chemical etching and removal of the photoresist from the etched structures, to the surface of the wafer was applied a layer of PMMA positive electron resist 0.3 μ thick, measured with an MII-4 microinterferometer. Selection of the PMMA's thickness was determined by the plasma chemical etching conditions. The electron resist was dried at 170 °C for 30 min.

A ZRM-12 unit was used for exposure. A combination of two rectangles having a 0.4 μ gap between them was selected as the exposure pattern. The length and width of the rectangles equaled 50 and 15 μ , respectively.

The wafer positioned on the carriage was adjusted in terms of vertical and horizontal axes with an error of ± 2 μ by means of a special manipulator. The exposure pattern was aligned with the structure's bridge semiautomatically from the control console.

Distances between structures along the vertical and horizontal axes were measured with precision of 0.1 μ on the ZRM-12 in the scanning electron microscope mode. The structure itself was used as the adjustment point and a new adjustment was made after every five exposed structures.

It is necessary to mention that it is possible to automate the entire exposure process when using special adjustment points and exposure programs. The accelerating voltage and the probe current during exposure equaled 25 kV and $2 \cdot 10^{-10}$ A, respectively. The time for point exposure of the pattern equaled 21 μs. The exposed structures were developed in MEK [methyl ethyl ketone] and IPS [isopropyl alcohol] for 30 s at a developer temperature of 23 °C.

Conclusion

A group technology has been developed for creating bridge-type Josephson junctions based on tin using a combination of optical and electron beam lithography processes and plasma chemical etching. In our opinion this route can be used for other materials, too, e.g., Nb.

Bibliography

1. Clarke, J. "Advances in SQUID Magnetometers," IEEE TRANS. ELECT. DEVICES, Vol 27, No 10, 1980, pp 1896-1908.

2. Ghecewala, T.R. "Josephson-Logic Devices and Circuit," Ibid., pp 1857-1869.

3. Zappe, H.H. "Memory-Cell Design in Josephson Technology," Ibid., pp 1870-1882.

4. Gamo, K. and Namba, S. "Josephson Junction Fabrication by Means of Ion Implantation and Electron Beam Lithography" in "Proc. 1st Conf. on Ion-Beam Modification of Materials," Budapest, 1978, pp 138-143.

5. Laibowitz, R.B. "Properties of Nb Josephson Microbridges," APPL. PHYS. LETT., Vol 23, No 7, 1973, pp 407-408.

6. Gu, J., Cha, W., Gamo, K. and Namba, S. "Properties of Niobium Superconducting Bridges Prepared by Electron-Beam Lithography and Ion Implantation," J. APPL. PHYS., Vol 50, No 10, 1979, pp 6437-6441.

8831
CSO: 1863/1

UDC 621.382.8

REPRODUCTION ACCURACY AND CHARACTERISTICS OF ION MICROLITHOGRAPHY RELIEF IMAGES

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 pp 61-63

[Article by Pyatras Pyatrovich Grigaytis, senior scientific associate, Polytechnical Institute, Kaunas]

[Text] Further development of computer hardware entails the mastery of new technological processes for fabricating large-scale integrated circuits and superlarge-scale integrated circuits [1], one of which is ion microlithography. The development of ion microlithography requires the solution of a number of problems, major among which are the development of a technique for producing controllable images in a resistive film, and determination of the characteristics of relief images.

Reproduction Accuracy

A high-resolution image formed by the electron beam lithography method is reproduced by the method shown in fig 1. A collimated stream of ions, 1, spatially modulated by a mask, 2, transfers the high-resolution image from the mask to a resistive film, 3, applied to a substrate, 4.

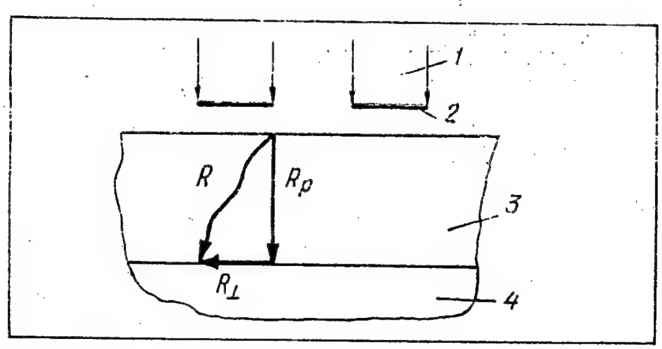


Figure 1. Method of Exposing Resist in Ion Microlithography

The ion stream, exposing the resist, interacts with the material of the resistive film and causes physical-chemical transformations in it, on whose basis a protective relief is formed which is used for creating images of elements on the substrate. The exposing ions are scattered in the resist's film with an average range of R (cf. fig 1). The scattered ions expose the resist's film in the area of the mask's geometrical shadow and their transverse range, $R_\perp$, determines the reproduction

accuracy of the ion microlithography method (the resist's film has a thickness equal to the projection range of the ions, R_p).

The mean range of ions interacting with a solid amorphous target can be determined by means of the Lindhard, Schiott and Scharf (LShSh) theory [2], by which it is possible to calculate the cross sections of interaction of incident ions with the target's material. This hampers application of the LShSh theory for technological calculations for ion microlithography.

In order to develop the exposed film of the resist, it is necessary that in the absorption of ions in it energy be released which is greater than the critical exposure dose. It is assumed that the exposing light ions ($z \leq 28$) undergo only electronic braking. This assumption makes it possible to use, for determining the magnitude of the mean range of ions, Bethe's energy loss equation for ions [3].

The expression for calculating the mean range of ions looks like this:

$$R = \frac{10^{-2} M J^2 (4\pi\varepsilon_0)^2}{32\pi e^4 z^2 Z^2 M_1 N} E_i \left(2 \ln \frac{4E_0}{J}\right), \tag{1}$$

where J is the mean energy of atomic transitions of the resist's material, equal to 65.5 eV; E_0 is the initial energy of the exposing ions; e is the charge of an electron; z is the atomic number of the exposing ion; Z is the effective charge of atoms of the resist's material; N is the concentration of the resist's atoms; $M = M_1M_2/M_1 + M_2$ is the reduced mass of the colliding system, where M_1 and M_2 represent the atomic masses of the two colliding systems (M_1 is the atomic mass of the incident particle); and E_i is the integral exponential function.

For the purpose of confirming the correctness of the assumption regarding electronic braking, calculations of the mean range of exposing ions according to expression (1) were compared with calculations of the same quantity according to the expression in [4]:

$$R' = \frac{(M_1 + M_2)^2 (z^{2/3} + Z^{2/3})}{10{,}7 a_0^2 N M_1 M_2} \rho(E),$$

where a_0 is the first Bohr radius and $\rho(E)$ is the relative range of an ion.

Function $\rho(E)$ was tabulated and for determining it quantity E was calculated:

$$E = \frac{4\pi\varepsilon_0 a_L M_2}{zZe^2 (M_1 + M_2)} E_0,$$

where a_L is the characteristic screening length according to Lindhard,

$$a_L = 0{,}885 a_0 (z^{2/3} + Z^{2/3})^{-1/2}.$$

Comparison of the results obtained when using Bethe's equation and the expression obtained from the LShSh theory demonstrated their good agreement. Therefore, it is suggested that Bethe's energy loss equation be used for technological calculations in ion microlithography.

The magnitude of the entry of exposing ions into the area of the mask's geometrical shadow is determined as follows

$$R_{\perp} = kR,$$

where k = 0.18, 0.26, 0.4 and 0.46 when exposing the resist with hydrogen, helium, argon and oxygen ions, respectively.

The results of calculating the accuracy of ion microlithography, determined according to the procedure suggested, are presented in table 1 for a number of exposing ions and their energy. As can be seen from this table, ion microlithography has high image reproduction accuracy, better than the reproduction accuracy of x-ray lithography [5].

Table 1. Reproduction Accuracy of Ion Microlithography, nm

1) Экспонирующие ионы	2) Энергия ионов, кэВ				
	50	100	200	300	500
H	22	90	320	659	1632
He	4	14	50	100	255
O	—	7	23	47	117
Ar	—	—	11	21	50

Key:
1. Exposing ions
2. Ion energy, keV

Relief Image Characteristics

The following are the key parameters of microlithography resistive films:

The threshold exposure dose, D_1, with which the insoluble negative-resist film begins to form and the positive-resist film begins to dissolve.

The critical exposure dose, D_2, in exposure to which the negative-resist film becomes completely insoluble and the positive-resist film is completely dissolved.

Sensitivity to the exposing radiation, $q = 1/D_2$.

The contrast factor, $\gamma = D_1/D_2$, which determines the resolution of the resistive material.

These parameters were determined from the characteristics of exposing negative (ELN-24) and positive (ELP-9) resistive materials. The resistive films were exposed to H^+ ion flux with energy of 50 keV and were developed for 20 s in standard developers at 20 °C. Relationships determined between the thickness of

the protective relief and the exposing radiation dose made it possible to determine the key parameters of resistive materials, which are summarized in table 2.

Table 2. Key Parameters of Resistive Materials Studied

Exposure	Material	D_1 ($\mu C/cm^2$)	D_2 ($\mu C/cm^2$)	q, ($cm^2/\mu C$)	γ
Ion	ELN-24	0.16	0.73	1.36	0.22
	ELP-9	0.10	0.92	1.08	0.108
Electron	ELN-24	0.47	10.0	0.1	0.047
	ELP-9	3.2	50.0	0.02	0.065

In electron lithography the critical exposure dose, D_2, for a negative and positive resistive material when exposing to electrons with energy of 30 keV equals 10 and 50 $\mu C/cm^2$, respectively.

As is obvious from table 2, the same resistive materials are more sensitive to ion exposure than they are to electron. In addition, the critical dose in exposing to an ion stream is approximately the same for both types of materials, which was mentioned in [6]. A high value of factor γ indicates the ability to reproduce with the resistive material patterns with high resolution.

A study was made of the resist properties of ion microlithography relief images in relation to the influence of a CF_2HCl plasma chemical discharge, using a radio-frequency plasma chemical etching system under the following conditions: gas pressure--5.33 Pa; power input--200 W; frequency--13.56 MHz; specimen temperature--180 °C.

The dissolution rates of the ELP-9 film and silicon dioxide differed by more than an order of magnitude, which testifies to the high resistive properties of ion microlithography relief images.

Conclusion

The procedure suggested for producing controllable planar images in resistive films, based on the Bethe energy loss equation for ions, made it possible to determine the reproduction accuracy of ion microlithography, which is better than the same parameter with x-ray lithography. Study of the characteristics of ion microlithography relief images showed greater sensitivity of resistive materials to ion exposure than to electron. Protective reliefs formed by ion microlithography have high resistance to plasma chemical etching.

Bibliography

1. Derkach, V.P. and Kukharchuk, M.S. "Electron Beam Lithography as an Effective Means for Mastery of Submicron Dimensions of Large-Scale Integrated Circuit Elements," MIKROELEKTRONIKA, Vol 9, No 6, 1980, pp 498-516.

2. Lindhard, J., Scharf, M. and Schiott, H. "Range Concepts and Heavy Ion Ranges (Not on Atomic Collisions)," MAT. FYS. MEDD. DAN. VID. SELSK., Vol 33, No 14, 1963, pp 1-42.

3. Mott, N. and Messi, G. "Teoriya atomnykh stolknoveniy" [Theory of Atomic Collisions], Moscow, Mir, 1969, 756 pages.

4. Pranyavichyus, L. and Dudonis, Yu. "Modifikatsiya svoystv tverdykh tel ionnymi puchkami" [Modification of Properties of Solids by Ion Beams], Vil'nyus, Mokslas, 1980, 242 pages.

5. Grigaytis, P.P. "Formation of Protective Reliefs by X-Ray Lithography Method" in "Fiziko-tekhnologicheskiye voprosy kibernetiki" [Physics and Technology Problems in Cybernetics], Kiev, IK AN USSR, 1979, pp 46-54.

6. Grigaitis, P. and Pranevičius, L. "Characteristics of the Relief Patterns in Ion-Beam Lithography," JAP. J. OF APPL. PHYS., Vol 20, No 4, 1981, pp L261-L263.

8831
CSO: 1863/1

YeS9009 TAPE CASSETTE DATA PREPARATION UNIT

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 inside back cover

[Item: "YeS9009 Tape Cassette Data Preparation Unit"]

[Text]

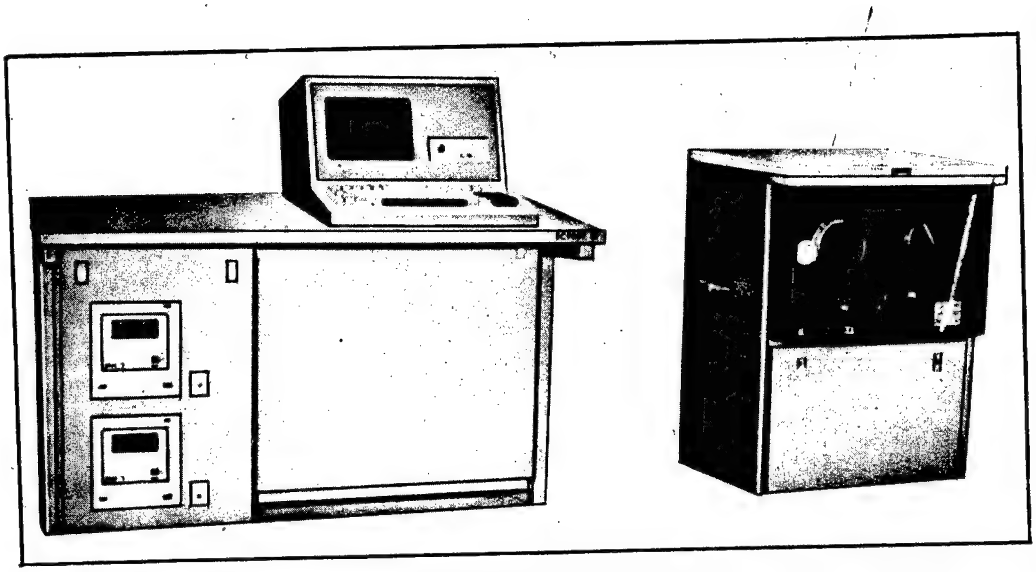


Figure 1. YeS9009 Unit

The YeS9009 is a self-contained programmable data preparation unit employing cassette magnetic tape with microprocessor control, whose use in place of traditional punches will make it possible to increase twofold the throughput of the data preparation process and lower scores of times data medium costs.

The unit possesses extensive functional capabilities and can enable the working process under the control of a user's program and without program control in the following modes:

Input of data from a keyboard onto cassette magnetic tape (3.81 mm).

Verification of data written on cassette magnetic tape.

Input of data with verification.

Search for and editing of data written on cassette magnetic tape.

Input of user programs into unit's on-line storage.

Translation of user programs from FOD (Format Description of Data) language into machine language.

Output of data from cassette magnetic tape to standard magnetic tape and for printout.

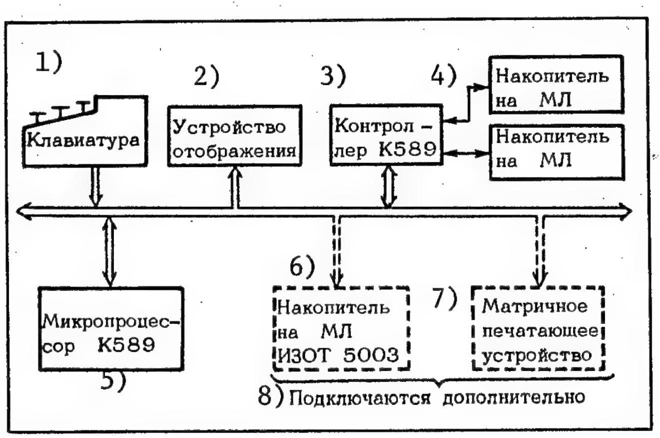
Recoding from DKOI [EBCDI] code into KOI-8 [data interchange code] code.

Diagnosis of malfunctions.

Programming is performed in the FOD user's language, making it possible to describe a large set of procedures for automating and controlling the process of keyboard input and preliminary preparation of data.

The element base of fourth-generation computers--large-scale integrated circuit microprocessing units and a semiconductor memory--are used in the unit's electronics.

A structural diagram of the unit is shown in fig 2.



Key:
1. Keyboard
2. Display
3. K589 controller
4. Tape storage
5. K589 microprocessor
6. IZOT 5003 tape storage
7. Matrix printer
8. Connected in addition

Figure 2. Structural Diagram of Unit

Composition and Technical Data

Control unit	Microcomputer (word length--8 bits, ROM capacity--2K bytes, RAM capacity--16K bytes)
Display	CRT (256 characters)
Keyboard	Alphanumeric (lower- and upper-case Russian and Roman alphabets, digits, special characters)
Tape storage	Ye5091 (two each)
Can be connected in addition	IZOT 5003 type miniature tape storage and YeS7934.02 matrix printer
Power requirement	0.8 kVA

8831
CSO: 1863/1

UDC 62-50.007

PRINCIPLES OF DESIGNING HIGHER LEVELS OF COMPUTER NETWORK SOFTWARE

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 6, Nov-Dec 81 (manuscript received 15 Jun 81, after revision 20 Aug 81) pp 3-12

[Article by A. I. Ilyushin, A. N. Myamlin and Vs. S. Shtarkman]

[Text] INTRODUCTION

One of the most serious problems in present-day computer networks is ensuring simplicity of inserting applied subsystems into the network arrangement, and also ensuring simplicity of using them. The resolution of this problem has a decisive effect on the success of introducing networks. In a recent paper, Dr. J. M. McQuillan points out, among other things, the failures of protocols of the highest level, and the absence of a simple versatile interface with the network for the host computer [Ref. 1]. The list of questions drawn up for further study on levels 7—4 in Ref. 2 shows that these problems have not been resolved even in the model of ISO architecture of open systems.

The unavailability of successful solutions for the enumerated problems leads to a plethora of network software in base computers in a volume that reaches hundreds of kilobytes in some realizations. This obviates connection of even average computers to the network, not to mention minicomputers and intelligent terminals.

In our opinion, problems of upper levels of computer networks cannot be satisfactorily resolved within the framework of the widespread ideology of conveying messages between processes, since this ideology is at odds with the principal mechanisms that are conventionally used for programming in the local case. The basis concept used for constructing programs in the local case is request for some operation that is realized either by "open" code substitution, or by request for a procedure. With transition to the distributed case, instead of the mechanism of request for *any* operation (as in the local case), the generally accepted approach assumes protocols of message exchange between remote processes that are *specific* to particular applications. As a result, the remote case does not reduce effectively to the local case, "transparency" of the network for the user is not ensured, and so on.

If the approach generally accepted at the present time for the remote case were to be taken for the local case, it would mean that there would be different

agreements on relations between the calling program and subprogram for intensionally different subprograms. As a result, for example, the subprogram of matrix inversion would be requested by the operator CALL1, and the subprogram of eigenvalue calculation would be requested by operator CALL2, etc. For the local case, such an approach seems strange, to say the least. For the remote case, the level of knowledge is such that the literature available to us makes practically no mention of approaches different from construction of special protocols for each application.

We have worked out a fundamentally new scheme of constructing upper levels that is based on an ideology of interaction of abstract objects (subsystems, clusters, procedure packages) [Ref. 3, 4]. The basis mechanism of this interaction is the mechanism of request for any procedure in the remote abstract object (subsystem) with transfer of both firmware and abstract objects as arguments and results. This mechanism is supported in the first place by a *single* upper-level protocol called a universal functional protocol (UFP), and secondly by agreements on the request for a procedure in the subsystem that are a direct expansion of the agreements existing in any operating system on relations between the calling program and subprogram. As a result, the following conversions are realized when a remote procedure is requested:

--conversion of the list of call parameters to a message sequence (on the part of the calling program) and the reverse conversion (on the part of the requested procedure);

--conversion of the results of operation of the procedure to a message sequence (on the part of the requested procedure) and the reverse conversion (on the part of the calling program).

Here from the standpoints of both the calling program and the requested procedure the network is logically "transparent". Specifically this means that the codes of applied programs coincide for the local and remote cases. For the local case, control is transferred directly to the requested procedure; for the remote case, control is transferred first to certain relational procedures, and only then to the requested procedure.

The 1976-1980 realization of the experimental SEKOP network [Ref. 5, 6] has confirmed some advantages of the given approach.

BASIC CONCEPTS. Our goal is to construct network software with consideration of the requirements of the upper level itself, where applied problems are solved. The starting point for such a construction is the idea of a model of some subject region functioning in one or more computers. The model is treated as a set of interacting *objects* (subsystems) arranged according to a target hierarchy and according to a hierarchy of representations [Ref. 7]. The presence of a hierarchy of representations is determined by the fact that each abstract object consists of a certain set of objects that is called the *representation* of this object, together with a set of procedures called a *cluster*. For example the representation of a file is a set of pages on a disk, while the cluster of the file is the set of access subprograms.

The target hierarchy can be given two interpretations. One is that this hierarchy corresponds to the most general structure of any subject region, and specifically: any object of the subject space consists of subobjects, these subobjects in turn consist of subsubobjects and so on (for example a file consists of entries, and the entries consist of fields). The other interpretation of the target hierarchy is a hierarchy of connections to subsystems during user work on the computer (e. g. when working with the terminal, the user first establishes contact with the operating system, then with the interactive editor, then with the specific textual file, and so on).

Let us return to the structure of the abstract object. The procedures of the object are used within the framework of processes that are considered totally conventional. Procedures of different objects, arranged in order of incoming requests, can be executed within a single process. The procedures of one object, and even a single procedure, can be executed simultaneously within the frameworks of several processes.

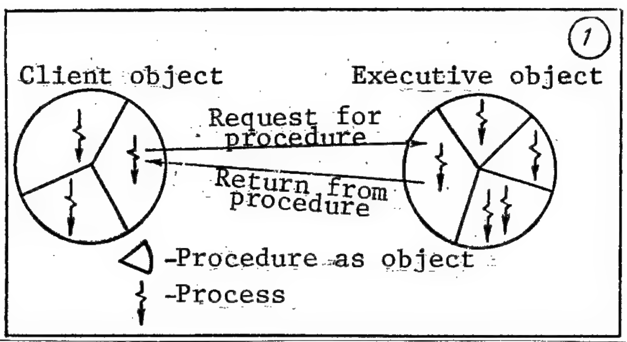


Diagram of elementary interaction between two objects

Objects interact by requesting procedures each in the other (Fig. 1). In this interaction, during the execution of a request the object in which the calling procedure is located is called the client object, and the object in which the requested procedure is located is called the executive object. Of course, either object may be simultaneously client and executive.

It is assumed that the requested procedure operates logically within the framework of the same process as the calling procedure, regardless of the fact that the client object and executive object may be located in different problems, or even in different computers. This means among other things that execution of the calling procedure within the framework of a given process is suspended during execution of a requested procedure. On the return, there are two alternatives: return with continuation of the work of the requested procedure as part of a logically new process; return with complete cessation of execution of the requested procedure. The term "logically new process" is used here because, for example, when the objects are located in different computers the requested program from the very beginning is, of course, executed within the framework of a process that is physically distinct from that within which the calling procedure is executed. Here the case of return with continued operation of the requested procedure is realized simply as continuation of the work of this physically new process which, in this case, becomes logically new as well.

Let us note one essential distinction of the request for a procedure in an executive object from the exchange of messages between processes: in the case of message exchange it is necessary that both sides agree to an exchange

("rendezvous"). In requesting a procedure it is assumed that entry of the executive object into a procedure necessitates only the "wish" of the calling procedure, and that this entry is accomplished within the framework of the same logical process in which the calling procedure operates. How the requested procedure responds is another matter. In particular, it may respond by refusing to execute the requested operation.

Let us compare the proposed model with that taken as the basis for the ISO model of open-system architecture [Ref. 2]. The ISO model assumes that there is a set of interacting applied processes on the applied level that exchange messages in accordance with protocols that are specific to the applied problems being solved. This postulates the necessity of developing a potentially infinite set of applied protocols.

In our opinion, such an idea of the applied level is incomplete, and is inadequate for constructing upper-level software. Here the concept of object (subsystem) and accordingly the concept of abstract type (cluster) do not suffice. We can readily see why these concepts are lacking in the ISO model. Conventional programming languages do not have them; the "send-receive" ideology predominates on the lower levels of network software. However, it has become clear in recent years that the concept of an abstract type of data is just as fundamental to programming as procedures. Where procedures are a means of realizing abstract operations realized in the host processor, clusters [Ref. 3] are a means of constructing abstract types of objects.

Clearly an abstract operation realized by a procedure requires abstract operands as well. However, unfortunately in most cases even now only procedures with arguments of built-in types are used in most cases in practical programming. Development of the ADA language under the auspices of the U. S. Department of Defense* brought hope that the situation would change. And it is not just that productive translators from ADA are showing up, but that in the offing is the possibility of using a "cluster-like" programming style in existing languages. It is in this way that we have used a macro-assembler and FORTRAN in the SEKOP network.

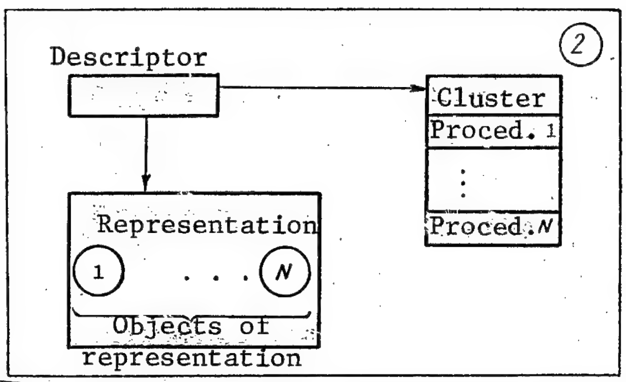
Introducing the concept of the abstract object (subsystem) into the model considerably alters the situation. The basis action becomes the request for operations in objects (request for procedures in subsystems) rather than the sending and receiving of messages. It becomes possible to write programs with consideration of abstract objects. In doing this, a certain operation, depending on the preceding tie-in, can be executed either as a request for a local procedure, or as a request for a procedure that is located in another computer. As an example of tie-in we can cite getting an object as a result of operation with an initiator subsystem. Another possibility is getting objects as call parameters.

Summing up, we can state that the acceptance of abstract types of data as one of the fundamental mechanisms of programming requires radical reexamination of the scheme for constructing network software.

*Nearly all programming languages that have appeared in the last few years have provisions for working with abstract types of data.

GENERAL SCHEME OF REALIZATION. The principal structures that will have to be considered in the realization are the scheme of the object (subsystem), objects of communication with remote objects, scheme of the request and execution of an operation in an object, scheme of transfer of arguments and return of results, and the scheme of creation of the object. Also playing an important part are the scheme of synchronization of objects, the scheme for handling malfunctions and the following restarts, and the scheme for realizing "indivisible" chains of operations.

Scheme of abstract object. The abstract object (subsystem) consists of a cluster, representation and descriptor of the object (Fig. 2).



Scheme of abstract object

The cluster is the set of procedures that execute the operations requested in the object. In this way the cluster determines the type of the object.

The representation is the set of objects corresponding to the type of object being represented, i. e. to the cluster. The objects that are part of the representation are accessible in all procedures of the cluster as the values of variables described in the title of the cluster, and called variables of representation.

The descriptor is a group of references to the cluster, to the representation, to the unique identifier of the object (UID), as well as other systems information.

Let us note that the scheme described above for construction of the object is not the only one possible. It is only necessary that the concept of the object be "materialized" in some way along with the concept of the procedure. It is desirable, of course, that work with objects be supported by an operating system. However, at present this is not the case, and therefore we will assume in the following that all the described facilities are supported by some monitor that we will call the monitor of objects (subsystems) and processes --MSP. Realization of the MSP depends considerably on the operating system that is used. We realized the MSP as an auxiliary job controlling the operation of user jobs.

An object is always created by another object. The creating and created objects, relative to one another, are called respectively the superobject and subobject, and the relation between them is a relation of subordination in the target hierarchy.

In creating an object, the superobject provides a cluster and a representation in the form of previously created objects. Matching of the cluster and the

representation comprises the intensional part of creating the object. There is also a systems, formal, part of creating the object that is executed by the MSP and consists of the descriptor, descriptor indicator and UID. The descriptor indicator is used for work with an object within the limits of one job, and it is this indicator that is assigned as the operand of operations working with the object. The internal structure of the indicator is known only to the MSP and is immaterial for applied programs. In the simplest case this is simply an address in the mathematical memory of the job.

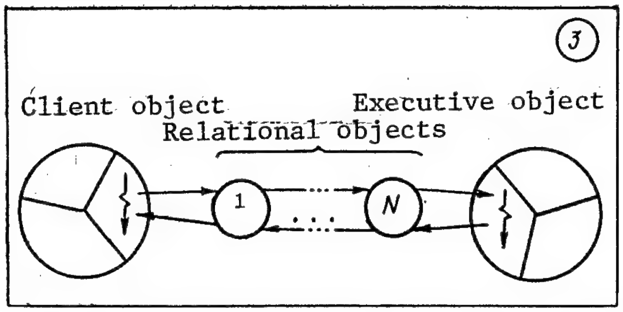
The UID uniquely identifies the object in a set of objects among which interaction may take place, and which may extend to all computers of the nation and exist over several years. For example, the UID may be realized as an aggregate of two numbers: the unique number of the computer and the astronomical time (unique within the limits of one computer) of creation of an object. UID's are used in the interaction of remote objects, i. e. objects that exist in different jobs, and possibly in different computers. All objects existing in a given computer are registered in a catalog of objects kept by the MSP.

The UID's are invisible in the user program. With objects in the body of some procedure, work proceeds both with values of local variables and formal parameters of the procedure, and with objects of representation. The formal parameters take the values upon request for this procedure. Values are assigned to local variables as a result of requests for operations. The variables of representation are given values either upon creation of an object analogously to the parameters of procedures, or as a result of requests for operations analogously to local procedures. Transfer of value objects is realized by transfer of their UID's.

The addressing of objects in the network by UID's is a very important feature of the proposed scheme. The fact is that the numbers of the clients and the numbers of the ports ordinarily used for addressing in the network are an analog of addresses of an immediate-access memory. Storage of references to objects in the form of addresses that do not uniquely identify an object and may be reused leads to difficulties in reorganizations, to "hanging" references and difficulties in restarts. UID's are only slightly more "costly" (48-64 characters) than addresses, and are free of these defects. The fact that UID's are used in the IBM System/38 on the level of machine operations gives a basis for believing that the overhead expenditures in working with them are low.

Relational objects. A considerable number of objects interact directly: in the calling procedure there is an indicator for the descriptor of the requested object that is also assigned in the primitive of the request. However, remote objects are accessible only via certain intermediate objects that will hereafter be called relational objects (Fig. 3).

The purpose of any relational object is to receive the request for an operation from a client object, to transport this request through some communication medium, request the procedure in the executive object, receive the return from the procedure of the executive object, transport it through the communication medium, and return finally to the calling procedure.



Request for operation
in remote object

The values of all argument parameters in a request from any relational object should coincide with those for the case of a direct request from the client object.

The values of all result parameters and the code of the response upon return from any relational object should coincide with those for the case of direct return from the executive object.

Of course, agreement in either case must be understood in some intensional sense because transformation of the representation is possible for example for different types of computers.

Thus relational objects are "transparent" both for the client object and for the executive object.

For each medium of communication, its own specific relational object is created. For example, for communication between computers a relational object may be created that uses protocol X.25 to support the request for operations in objects that are situated in remote computers.

Although there may be sequentially connected relational objects in the general case between two intensional objects, in the following we will consider the case with only one relational object.

We will give the following classification of requests for operations in objects:

P1--direct request within the limits of one physical process;

Z1--request through relational object within the limits of one job. The requested procedure works within the framework of a process physically distinct from the calling process. [This request] is used for example for initial debugging of large systems or for organizing parallel execution of operations;

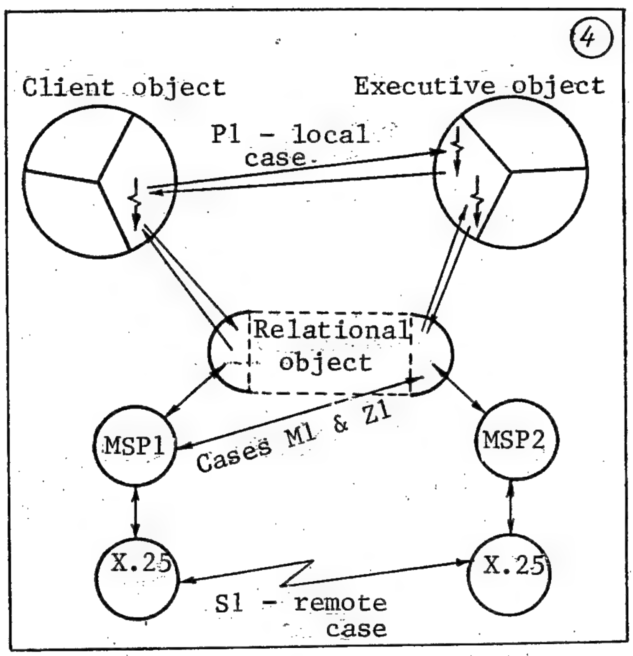
M1--request via relational object within the limits of one computer. The requested object is in a problem distinct from that in which the client object is located. Solution of some problem by simultaneous execution of several jobs on one computer is frequently used in cases where dynamic interrogation of resources is not possible within the limits of one job, or where there are limitations on the size of the resources singled out for one problem, or finally, where it is impossible to shield processes from one another within the limits of one job;

S1--request via relational object within the limits of one computer network. The requested object is in a computer distinct from that in which the client object is located.

Cases Z1, M1 and S1 are cases of request for an operation in the remote object. Each of these cases uses a relational object specific to the medium of communication between objects. In case P1 there may be no relational objects.

Although in cases Z1, M1, S1 the requested procedure always operates within the framework of a process physically distinct from the calling process, nevertheless the requested and calling processes are treated as parts of one logical process.

Scheme of request for operation. In any case (P1, Z1, M1, S1) control is transferred from the calling procedure to the cluster with respect to the descriptor indicator. If this is a target executive cluster, after this the execution of the target procedure that realizes the operation begins (Fig. 4). If this is a relational cluster, it selects from the descriptor of the relational object the UID of the target object, and calls the MSP.



Scheme of request for operation

The MSP generates the UID of the request, which uniquely identifies the request, like the UID of the object, and then determines where the object is located from the UID of the object (e. g. from the computer number).

In the case of a local computer (Z1, M1) the MSP seeks the object in accordance with the UID in its catalog of objects. In the case of a remote computer (S1) the local MSP transmits request parameters through the communication network to the remote MSP. Then the remote MSP looks up the object according to the UID in its own catalog of objects.

If the UID of the object is found, the MSP sets the UID of the request in line with the UID of the object. When the UID of the request is first in line of the UID's waiting for entry to the object, the MSP through, so to speak, the second physical half from the logical standpoint of the relational object, transfers control to the procedure of the target executive object. The goal procedure is executed as if this were a local request ("transparency"

of the relational object), and control is returned to the second half of the relational object. The relational object calls up the MSP, which removes the UID of the request from the line to the UID of the object, and in case S1 transfers the results of the request, using the UID of the request as the return address, to the initial MSP. This is followed by dispatch of the results to the calling job, return to the first half of the relational object, and finally, return to the calling procedure.

In the case of an accident with the executive object, the MSP removes its UID, and all request UID's standing in line to it, dispatching to client objects the results indicating emergency termination of the request.

Thus the following scheme of interaction of objects may be realized in a distributed network. The communication network operating for example on the X.25 standard provides a virtual channel between each pair of host computers (actually between MSP's). The MSP in the host computer provides communication with all objects inside it with addressing by UID's. The same scheme is proposed for direct connection to a network coupling minicomputers and intelligent terminals based on microprocessors.

Scheme of transferring arguments and returning results. In order for the MSP to be able to pack arguments and results of the request for operations into messages transmitted through the communication network, it requires descriptors of parameters. The descriptors are also necessary for converting the representation of parameter values in the case of different types of computers and dynamic verification of types.

The descriptor must primarily indicate: the type of parameter (argument, result, universal) where the value is situated (the address available in the descriptor is indicated directly after the descriptor or on it), and the type of value. The following types of values are possible:

--abstract object;
--scalars of built-in types (boolean, whole-number, real, line, etc.);
--data block of similar values (the block descriptor includes the number of elements of the block and an indicator on the descriptor of the value in which there is the type of element of the block, and which indicates the block of values itself);
--an entry that consists of different types of values (the entry descriptor has the number of fields in the entry and an index to the block of descriptors that describe the fields).

Also permitted are combinations like block of entries, entry of blocks, etc.

Let us consider how parameters are transferred on a request.

With a local request (case P1), control is transferred to the target cluster directly from the calling procedure. Here, in accordance with agreements on relations between the program and subprogram in one of the communication registers there will be an address of the region of the state of the requested procedure with a block descriptor of actual parameters. Then normal operation

of the requested procedure is possible. In case ZI, the region of the state of the requested procedure must be formed in the stack of a new process. Since both processes are in the memory of one job, transfer of parameters by reference is possible. In cases Ml and Sl, transfer of parameters is possible only by value.

Let us consider execution of a remote request in case Sl. Control in this case falls to the MSP, which begins to look through the block of descriptors of actual parameters. In accordance with the type of parameter and the type of value, the MSP selects values of parameters and packs them together with the descriptors into bursts for transmission through the communication network to a remote MSP. The volume of the bursts between MSP's is regulated by the universal functional protocol (UFP). In particular, the UFP specifies that the UID of the requested object and the name (number) of the requested operation are in the first burst. Upon receiving the first burst, the remote MSP finds the target object from the UID through the list of objects, and obtains the address of the list of formal parameters from the target cluster using the name (number) of the operation.

It is assumed that the descriptors of formal parameters define the regions of memory that are set aside (e. g. in the stack of the requested procedure) for values of the transmitted parameters. The MSP unpacks the received bursts, transcribing the values of the parameters from the bursts into the sections of memory set aside. During unpacking, a check is made on correspondence between types of actual and formal parameters. Naturally the unpacking of values proceeds in accordance with the type of parameter and the type of value. After preparing all parameters of the request, the MSP transfers control to the target cluster. After returning from the requested procedure, the MSP, guided by descriptors of parameters, packs the value of results into bursts and sends them to the initial MSP. The initial MSP unpacks the bursts with results and, after distributing the results, returns control to the relational object and thereby to the calling procedure. Let us note some peculiarities of transmission of parameters:

--blocks, like scalars, are packed into bursts and distributed to the other side;
--transformation of the representation is possible in the packing-unpacking of the value in the case of different types of computers;
--in the case of an abstract object, the UID of this object that is sent is taken from its descriptor. In unpacking, a relational object is formed whose descriptor contains the transmitted UID.

The scheme presented above for description of parameters was used for the BESM-6 computer. Let us note that the suggested general scheme of network software construction does not require strict observance of the presented scheme of description of parameters. The only important thing is the availability of some scheme defined for the given computer that enables the MSP to carry out the necessary transformations.

Now let us consider the question of where the descriptors themselves are taken from. Obviously the existing compilers from programming languages do not

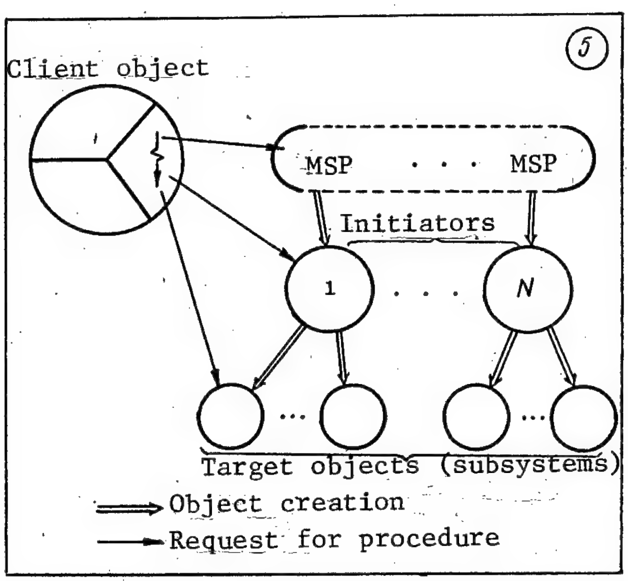
generate them, at least in the described form. We used macroassembler [Ref. 8] and FORTRAN. Macrodefinitions for the procedure title, the procedure request, the cluster title and so on were written in macroassembler. The procedure and cluster titles could describe types of formal parameters and types of local quantities from which the corresponding descriptors are generated. In FORTRAN, all descriptors have to be generated by the request for the corresponding subprograms.

It seems to us from our work experience with the proposed scheme that two conclusions can be drawn:

1) practical work is completely possible with existing languages and compilers;

2) the proposed scheme in future will require languages of the ADA type, the corresponding compilers and operating systems.

Scheme of object creation. Let a client object wish to create some executive object. As has already been stated, the intensional part of the creation process (matching the cluster and objects of representation) is executed by a superobject of the target executive object, while the formal part is executed by the MSP upon command of the superobject. From the standpoint of the client object, there has only been a request for the corresponding operation in the superobject. The result of this operation is an index to a descriptor (of either the target object or a relational object) incorporated in some local variable. It is assumed that the superobject has previously become accessible to the client. The very "first" object that can be considered always accessible is the MSP that has operations of connection to an initiator object of subsystems (objects). The subsystem initiator can start any subsystem stored in the local data bank. Although only the local MSP is accessible, it should be considered that the MSP's on all computers connected by the communication network logically comprise a single unit. This means specifically that connections can be made via the MSP to the initiator on any computer that is part of the network (Fig. 5).



Scheme of object creation

An object once created can be transferred to another object as an argument, as a result, and as an object of representation.

INDIVIDUAL PROBLEMS. Unfortunately, the scope of a journal article does not afford space for the scheme

of object synchronization, the scheme for organizing "restarts" and "indivisible" chains of operations. However, we hope that even the data presented here will give an idea of the general scheme of realization of upper levels of network software. Let us turn now to comparing the proposed scheme with the well-known ISO model [Ref. 2]. In the presented scheme the "applied level" (level 7) of this model branches into two parts. One part, relating to the request for operations in arbitrary subsystems (objects) is standardized. The other part, relating to the set of operations itself in the subsystems, and to the parameters of access to them, is nonspecific to the network. This is simply programming in languages with abstract types of data, irrespective of the local or distributed case.

In particular, it is nonspecific to the network that what is usually termed "establishment of connection" is a simple request for the operation of connection in the corresponding initiator. The result of such an operation is either a local or a remote object. Let us emphasize that even an initiator has no network specifics.

Thus the uppermost level of network software is the level that we have called the level of remote request for procedures in subsystems (LRR). LRR programs are located partly in MSP's and partly in user jobs. It seems to us that level 3 of the ISO model should be below the LRR. In the first version of the SEKOP network, the LRR utilized the facilities of the transport level (level 4). At the present time a version is being realized with output directly to level 3.

Preliminary results show that X.25 facilities are completely sufficient for realization of LRR. Strange as it may seem, apparently the elimination of level 4 will reduce the software on level LRR (not to mention the absence of software for level 4 itself).

Thus in the proposed model there are only four levels of systems network softwave. The entire applied software, including systems that support input of assignments, control of files, virtual terminals, will become identical for the local and distributed cases.

The question may arise as to whether the LRR will be too cumbersome since it lumps together all problems of levels 7—4 of the ISO model. It turns out that certain problems completely vanish, while others are much more simply resolved since it becomes possible in their solution to use information, so to speak, of the uppermost level. Let us consider some examples.

Flow control. Consider the problem of flow control in request for a remote procedure. If the arguments or results are large blocks, it would seem that problems of buffering would arise. However, after reception of the first burst of the request with the list of descriptors of actual parameters on the executive subsystem side, types will be verified and a place will be set aside in the job memory for local variables of the requested procedure (including for the values of parameters). No problems can come up that involve lack of space for values that are sent in subsequent copying of parameters from the store of the calling job to the store of the requested job.

Transformation of representation in case of different computer types. Consider first the problem of transforming a representation for values of built-in types. Clearly, given the value descriptors, the LRR can in most cases transform, e. g. an integer to an integer, or change the coding of symbols. Considering that there are not many built-in types or computer types, this is a completely realistic job. However, the question may arise as to how, for example, a 32-bit number of a large computer is transformed to a 16-bit integer of a minicomputer. The answer here is very simple. If the requested subsystem is in the 16-bit computer, it can operate intensionally only with the range of integers determined by 16 digits. Consequently, assignment of an integer that goes beyond this range as a parameter is an intensional error (of the same kind as, for example, assigning a number greater than 80 or 132 as the length of a line for output to a terminal). And so the LRR performs the transformation if possible. Otherwise the request for an operation is terminated as an emergency case with respect to a check on types of parameters.

Consider now the transformation of a representation of more complicated types of data based on the example of file copying. A widely known approach [Ref. 2] suggests standardization of some virtual file repository. Then on each side, the network services of level 6 transform the representation of data that is used in the special repository of files to the standardized representation and vice versa. It seems to us that such an approach is completely unsatisfactory since it violates the principle of "screening" of information with all the attendant consequences. In particular, the network services become sensitive to changes in the internal structure of the particular data control systems. Besides, the very possibility of working out a satisfactory universal file repository is suspect, since efforts at standardization as a rule extend over many years. And during this time, the intensional concepts themselves in the region being standardized usually undergo considerable changes. It is well known that perhaps the only real result of efforts to develop, say, a virtual terminal (or network copying of assignments) has been the introduction of many classes of virtual terminals (and accordingly, classes of assignment copying), i. e. the region of research has been expanded without getting any results of practical importance.

In the scheme that we have proposed, the problem is resolved in another way. The internal structure of the file considered as an abstract object is taken as fundamentally inaccessible. Copying is done for example by a utility routine that operates with two subsystems: local and remote file repositories, using their operations that are locally accessible in the usual way. The utility routine must know the set of fields in the entries of the file to be copied, and the types of their values. Then the entries are copied by the simplest "read-write" cycle. In doing this, the particular data control systems themselves handle transformation from the internal representation to the entry located in the permanent store and back. The LRR handles transformation of a value of the built-in type to entries.

Thus the problem of transforming the representation of structured data reduces to transformation of the values of arguments and results of built-in types. On the other hand, conversion from an internal representation to arguments and results of built-in types is handled by the particular subsystems themselves.

Of course, one problem remains. Although from the standpoint of request techniques it is immaterial for the calling program whether the operations being requested are in the local or remote subsystem, it must know the particular sets of operations in each subsystem that is used. However, even here it is possible in many cases to do without "forced" standardization by using interface structures. Let us recall that programming is done "from the top down". The program of the uppermost level is written with regard to abstract objects of the next lower level of representation. In doing this, it is assumed that these abstract objects are "ideally" suited to solution of the problem at hand.

The next step is programming of clusters for "ideal" objects. One such programming method may be selection of certain clusters (subsystems) that are already available. To eliminate the difference between "ideal" objects and those already available, certain interfacing clusters must be written. Thus the need for writing interface clusters is already present in the method of programming "from the top down". Of course, it is all well and good if some standard has naturally come about in some region. However, even without a standard, it is fairly simple to operate in the given arrangement without resorting to "forced" standardization of the network file repository type.

Work with "old" systems. In any scheme of realization claiming practical feasibility, a solution must be found for the problem of working with previously written local applied subsystems. When such subsystems are inserted into the network arrangement, we distinguish two interfaces: "upward" and "downward". By an "upward" interface we mean one between the old subsystem and the programs that use it (for example the interface between the controlling program IMS and applied programs). By "downward" interface we mean one between the old subsystem and the subsystems used by it (for example the interface between IMS and terminals).

To realize an "upward" interface, it is proposed that an interface cluster be written for each old subsystem, after which the subsystem becomes accessible at any point of the network. Judging from our experience, writing interface clusters necessitates small expenditures. For example, creation of interface clusters for a local system of starting assignments took a few weeks.

As to the "downward" interface, it seems to us that it can be assumed in most cases that old subsystems use only real I/O devices "downward". Then the downward interface can be situated at the level of the standard hardware interface between the I/O channel and the I/O device. Connected to the channel in place of the standard I/O device is an emulating processor that represents a "real" (from the standpoint of the old subsystem) I/O device to any required network unit. This approach has been successfully used for terminals in the BESM-6.

ISO model. A considerable problem that differs in nature from those considered in the preceding sections is relation to the ISO model, and in particular to its levels 7—4. It seems to us that the question is fairly clear from the scientific and engineering standpoint. The ISO model is at base unsatisfactory in the part relating to levels 7—4. This is clear in particular from

the fact that the authors of the model were unable themselves in Ref. 2 to resolve the protocols of the virtual terminal, copying of assignments and copying of files with respect to levels of the model. In the proposed four-level model, problems are resolved much more simply, although it is easy to imagine "political" difficulties to its introduction.

CONCLUSION. We have considered here a scheme for constructing network software as an alternative to that on levels 7—4 of the ISO model. The main difference is that the proposed scheme is based on interaction of abstract objects (subsystems) accomplished by request for procedures in these objects, whereas the scheme of the ISO model is based on interaction of processes that is realized by sending and receiving messages. In the proposed model, above the third level is a level of remote request of subsystems that is the last level in the systems network software. In other words, the applied level actually has no network specifics, and in most cases reduces effectively to local applied software. Realization of upper levels of the SEKOP network [Ref. 5] has confirmed some advantages of the proposed approach:

1. Bilateral transparency of the network is ensured. On the one hand, the user of subsystems requests operations in them irrespective of the placement of these subsystems relative to this user -- local or remote. On the other hand, the subsystems (procedures in a subsystem) do not depend on how they are requested -- locally or remotely. Thus a simple interface can be assured between the users and makers of the subsystems in the form of agreements about relations when requesting procedures. Here the local case is *effectively* realized.

2. Expenditures on creating applied subsystems are reduced since the development of specific protocols is eliminated. Here distributed programming actually reduces to the local case. In particular, in setting up the SEKOP network, the realization of, say, copying of assignments has actually been reduced to writing interface clusters for a local system of starting assignments and took a few weeks as contrasted with several years usually required for developing and realizing conventional protocols.

3. The network software logic becomes more structured and transparent.

4. Transformations of data representation can be naturally reduced to transformation of built-in types.

5. The universal protocol does not give rise to excess messages or worsening of response time.

6. Agreement and standardization is required only for one universal functional protocol (standards on levels 1—3 have basically been combined). For purposes of comparison, let us point out that the ISO model requires standardization of the protocols of levels 4 and 5, and also a potentially infinite number of protocols on levels 6 and 7.

REFERENCES

1. McQuillan, J. M., "Local Network Technology and the Lessons of History", COMPUTER NETWORKS, Vol 4, No 5, 1980, pp 235-238.

2. "Reference Model of Open System Interconnection", ISO/TC 97/SC16, No 227, 182 pp.

3. Liskov, B., Snyder, A., Atkinson, R. et al., "Abstract Mechanisms in CLU", COMMUNS ACM, Vol 20, No 8, 1977, pp 564-576.

4. Wegner, P., "Programming With ADA", New York, Prentice-Hall, 1980, 212 pp.

5. Drozhzhinov, V. I., Ilyushin, A. I., Myamlin, A. N., Shtarkman, Vs. S., "Principles of Designing the Experimental SEKOP Multiple-User Computer Network", VOPROSY KIBERNETIKI, No 57, 1979, "Problemy informatsionnogo obmena v vychislitel'nykh setyakh" [Problems of Information Exchange in Computer Networks], pp 18-33.

6. Ilyushin, A. I., Shtarkman, Vs. S., "Method of Constructing Applied Level of Computer Network Software", PROGRAMMIROVANIYE, No 6, 1979, pp 34-43.

7. Ilyushin, A. I., Filippov, V. I., "Multilevel Model of Database and Information Retrieval System Architecture", PROGRAMMIROVANIYE, No 6, 1980, pp 64-71.

8. Mikhelev, V. M., Shtarkman, V. S., "MAKROKOD - opisaniye yazyka" [MAKROKOD - Description of Language], Moscow, 1972, 50 pp (Preprint No 24, "Institute of Problems of Mechanics, USSR Academy of Sciences).

6610
CSO: 1863/16

UDC 681.324

DEVELOPMENT OF TERMINAL NETWORK ACCESS METHOD FOR YeS OPERATING SYSTEM

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 6, Nov-Dec 81 (manuscript received 12 May 81) pp 12-16

[Article by I. L. Dayen]

[Text] INTRODUCTION. In developing terminal-oriented programs for a computer network, it is often necessary to consider the diversity of peripheral hardware. For example a network may use the YeS7906, VT-340, T-63, AP-70 terminals and the like that interact with such systems as the SRV [time-sharing system] or DUVZ [expansion not given] on YeS EVM computers, or with the DIMON system on BESM-6 computers. As a rule, interface modules must be made in a corresponding number for serving terminals of different types by the programming system.

However, terminals of different types can be said to be similar, enabling us to define a standard representation for a certain set of terminals, and a standard protocol of terminal-program interaction. It is convenient to divide the set of all terminals into classes, where each class of developed terminals subsumes a class of simpler devices. Such an approach necessitates making modules both on the part of the terminal, and on the part of the program for converting the former method of interaction to the standard protocol. Clearly with an increase in diversity of peripheral hardware, an approach based on a standard representation of terminals is more preferable.

The standard method of terminal representation is usually called a *virtual terminal (VT) of the network* [Ref. 1]. The protocol that regulates the interaction of a program and a virtual terminal is called the *virtual terminal protocol (VTP)*. It is because such interaction is based on unified agreements about formats and the sequence of exchange of transactions that the program "sees" different terminals as some standard model that is adjusted to a specific profile within a class depending on the characteristics of the peripheral hardware.

Macrocommands of the *terminal network access method (TNAM)* that realizes the VTP serve as the means of developing programs for servicing terminals in the network medium. A TNAM program called the virtual terminal emulator performs the functions of mapping the real terminal onto the VR, and also generating and interpreting VTP commands. Each type of terminal requires programming of its own emulator.

At present there is no international standard for VTP's, although concerted action is being taken in this direction by the ISO [Ref. 2] and CCITT [Ref. 3]. A detailed analysis of trends in the field of VTP development can be found in Ref. 4. Interesting versions of VTP's have been realized in the EIN, EURONET and BUN networks [Ref. 5, 6]. The VTP version developed here is based on experience with the above-mentioned networks. No limitations are imposed on methods of connecting terminals to a computer network. A TNAM has been developed for the YeS operating system based on the network access method of Ref. 7, the core being a transport station that realizes INWG-96 protocol. As of now, an emulator has been developed for the YeS7096 display complex, and work is also being done on using other peripheral hardware.

FUNCTIONS OF VIRTUAL TERMINAL PROTOCOL. The VTP regulates interaction over a virtual connection. Two phases are distinguished in the VTP: a phase of representation control during which the profile of the virtual terminal is adjusted, and a data transport phase.

On the representation control phase, the program as a rule interrogates the range of some (or all) parameters of the virtual terminal, and selects the required VT profile based on analysis of the list of parameters.

Access may be had to the information mapping device from either end of the virtual connection. The YOUR TURN tag incorporated into the transaction serves as a means of synchronizing access. Upon receiving this tag, the passive partner is switched to the active state. This form of dialog is called two-way transmission by turns. A free mode of exchanging transactions is also possible in the VTP.

In implementing terminal-oriented interaction, the text is represented in structural form: the data have the structure of pages, lines, bit positions.

On the data transport phase the transactions are wiped out, which is accomplished by "dropping" (clearing) the connection. Even in this case, interpretation of dialog control commands must continue.

Facilities furnished by the network access method are used for sending an interrupt signal (telegram).

BRIEF DESCRIPTION OF VIRTUAL TERMINAL. From the program standpoint, the virtual terminal consists of the following components: information mapping device, keyboard, connection control device, control device.

The information mapping device is a two-dimensional data structure that corresponds to the number of symbols in a line (X) and the number of lines in a page (Y). Each position contains either a symbol in KOI-7 code or a field attribute (see below) or else a nonsignificant zero (X'00').

Each position within the data structure is characterized by a coordinate pair (x, y): $0 \leqslant x \leqslant X$, $0 \leqslant y \leqslant Y$. Coordinates of the instantaneous position are given to the address index. The value of the address index may not coincide with the instantaneous position of the cursor on the actual imager.

Characteristic of VT	Classes of VT's		
	Line	Page	With data processing
Addressing:			
Following symbol (by omission)	X	X	X
New line (by omission)	X	X	X
Carriage return	X	X	X
Absolute addressing	X	X	X
Relative addressing	X	X	X
Next unprotected field	—	—	X
Field attributes:			
A_1--protected/unprotected	—	—	X
A_2--digital/alphanumeric	—	—	X
A_3--field of selection determined by photoselector	—	—	X
A_4--field of interruption determined by photoselector	—	—	X
V_1--mapped/unmapped	—	—	X
V_2--normal contrast/enhanced contrast	—	—	X
Clearing of data structure	—	X	X
Initialization of data structure	—	—	X
Symbols: — characteristic not used X characteristic used			

The VT data structure is broken down into fields that have different attributes of access control and contrast.

The process of data recording in the VT is a sequential process that is always associated with a change in value of the address index. Process sequence is understood in the sense that once access has been gained to the mapping device, none of the partners on a connection yields the right of access until the recording process has been completed.

A change in value of the address index can be initiated by VTP addressing commands without modifying the content of the data structure.

The original data structure is established on the phase of representation control, and may then be modified on the data transport phase. The VT data structure may not correspond to the physical characteristics of the imager.

The keyboard is a data input device. Keys are classified as follows:

--alphanumeric,
--addressing,
--functional,
--keys for interrupt signal input,
--keys for initializing and clearing VT data structure,
--INPUT keys,
--photoselector,
--keys for local editing.

The connection control device serves for control of establishing virtual connection and for breaking it.

The control device realizes the VTP logic.

The following three classes of virtual terminals are distinguished:

--line VT ($X \geq 0$),
--page VT ($X > 0$, $Y > 0$),
--VT with data processing ($X > 0$, $Y > 0$) permitting breakdown of the data structure into fields with predetermined attributes.

Permissible characteristics of virtual terminals of different classes are summarized in the table on the preceding page.

MACROCOMMANDS OF TERMINAL NETWORK ACCESS METHOD. The controlling program of the TNAM has been constructed as an expansion of programs of the network access method and includes a dispatcher with internal processes. The TNAM is an analog of methods of access with queues of the YeS operating system. The applied programmer must reserve storage for input and output buffers that may contain several VTP commands.

List of macrocommands:
INTLOGON - connect to TNAM;
INTLISTN - listen to connection (partner ready to establish connection);
INTDIAL - establish connection;
INTPUT - form DATA command (symbol sequence);
INTGET - receive command;
INTYRTRN - reverse virtual connection upon initiative of active partner (incorporation of YOUR TURN tag);
INTPLSE - request to reverse virtual connection on the part of passive partner;
INTFNCN - FUNCTION command (e. g. response to function key);
INTADDR - addressing command;
INTNEWFD - form field with given attributes;
INTERASE - clear VT data structure with retention of breakdown into fields;
INTINIT - initiate VT data structure;
INTINT - send interrupt signal;
INTTRUNC - truncate buffer;
INTRELSE - release buffer;
INTHIDE - secret input (data fed in with zero contrast);

INTCLEAR - "drop" virtual connection;
INTQUIT - break connection;
INTLOGFF - disconnecte from TNAM.

A simple example is given below of programming in the terminal network access method.

EXAMPLE

```
                        .
                        .
                        .
                       INTLOGON SNCB, 5 - connect to TNAM
*5 - number of resources in connection (BUFNOIN+BUFNOUT+1)
                        .
                        .
                        .
             INTLISTN ASCB,VTCD,6,SNCB, - listen to connection
                       BUFIN=IN,BUFLIN=120, BUFNOIN=2
                       BUFOUT=OUT, BUFLOUT=120, BUFNOUT=2
*ASCB - expanded channel block in VTP mode
*M    - establish semiduplex mode, go to active state
*VTCD - interaction in VT code
*6    - number of the local port
*BUFLIN,BUFNOIN(BUFLOUT,BUFNOUT)          - size and number of input (output)
                                            buffers
                       WAIT    ECB=ASCB+X'1BO' - wait to establish connection
SEND                   .
                       .
                       .
                       INTPUT ASCB,AREA1       - position recording
                       WAIT    ECB=ASCB+X'16C' - go to passove state
RECVE                  INTGET ASCB,AREA2       - obtain recording
                       .                       - process input entries
                       .
                       .
                       TM      AREA2, X'10'    - is YOUR TURN tag established?
                       BO      SEND            - yes, tag established
                       B       RECVE           - continue reading
                       .
                       .
                       .
SNCB                   DC      242F'0'         - block connected in VTP mode
IN                     DC      2CL120'-'       - pooled input buffer
OUT                    DC      2CL'120'-'      - pooled output buffer
AREA1                  DC      F'0',CL100'-'   - working area 1
AREA2                  DC      F'0',CL100'-'   - working area 2
```

EMULATOR OF VIRTUAL TERMINALS (FOR YeS7906). The command language of the emulator consists of the following commands [Ref. 8]:

LOGON - connect to network;
LOGOFF - disconnect from network;

DIAL - establish connection with predetermined port;
QUIT -- break connection.

After input of the LOGON command, the emulator is in the "listening" state, i. e. waiting for access from the network. This state may be noted by the DIAL command initiating active establishment of connection. When connection has been established, the operator may introduce information from the terminal and obtain transactions from a remote program. Only the two-way mode of transaction exchange by turns is permitted. If an operator is active, an invitational symbol will be displayed on the screen, and the emulator is ready to receive data from the terminal. After data input by the operator, the keyboard is blocked and the initiative goes to the program. In the passive state, an ATTENTION signal reverses the connection.

The VT modulator may be a remote program. Therefore we may say that the set of emulators forms a terminal network service that enables interaction among terminal operators.

TNAM macrocommands were used in programming the emulator.

DIRECTION OF FUTURE WORK. Plans call for:

--making emulators primarily for the YeS7920 display complex and the YeS7070 printer;

--developing a terminal processor based on a series-produced minicomputer with network VTP interfacing. Such a processor should perform the functions of a terminal concentrator, and become a comparatively inexpensive device for ensuring access of users to network services;

--making DUVZ and SRV packages for VTP's enabling one to take advantage of computer operation in the network mode;

At present the terminal network access method is in experimental use.

REFERENCES

1. Davis, D., Barber, D., "Computer Communication Networks", Moscow, Mir, 1976, 680 pp.

2. International Standard Organization. Title: "Proposal for a Standard Virtual Terminal Protocol" in: "Computer Network Protocol Symposium, Liege, 1978", pp H27-H49.

3. CCITT Proposals for Draft Provisional Recommendations for Interworking Between Non-Packet Mode and Packet-Mode DTE -- CCITT Study Group VII. Temporary Document 62-E. Geneva, 1977, 62 pp.

4. Magnee, F., Endrizzi, A., Day, J., "A Survey of Terminal Protocols", COMPUTER NETWORKS, Vol 3, No 5, 1979, pp 299-314.

5. Schichter, P., Zimmermann, H., "Proposal for a Scroll Mode Virtual Terminal", COMPUTER COMMUNICATIONS REVIEW, Vol 7, No 3, 1977, pp 23-55.

6. Bauwens, E., Magnee, F., "The Virtual Terminal Approach in the Belgian University Network", COMPUTER NETWORKS, Vol 2, No 4/5, 1978, pp 297-311.

7. Gusev, V. V., Nikolenko, D. I., Petrukhina, L. V., "Principal Macrocommands of Network Access Method" in: "Vychislitel'nyye seti kommutatsii paketov: Tezisy dokladov" [Packet-Mode Computer Networks: Abstracts of Papers], Riga, 1979, pp 151-154.

8. Dayen, I. L., Kavitskiy, S. B., Ozirnyy, V. L., "Realization of Some Function-Oriented Computer Network Protocols for the YeS Operating System" in: "Vychislitel'nyye sistemy, seti i tsentry kollektivnogo pol'zovaniya" [Computer Systems, Networks and Collective-User Centers], Abstracts of Scientific Reports to All-Union Conference in Novosibirsk, 1978, Part 3, pp 235-238.

6610
CSO: 1863/16

UDC 681.3.069

APPROACH TO LOAD FACTOR MEASUREMENT FOR REAL-TIME COMPUTER SYSTEM IN DOS MEDIUM BASED ON AGGREGATED SOFTWARE SYSTEM (ASPO DOS)

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 6, Nov-Dec 81 (manuscript received 4 May 81) pp 93-96

[Article by N. N. Merenkov and A. P. Popov]

[Text] The need for measurement of a real-time (RT) computer system is dictated by the necessity of getting information on dynamic characteristics of the hardware-software complex to analyze and improve the quality of operation of the system [Ref. 1-3].

The main difficulties that arise in using classical methods of measurements of a computer system (interception and sampling of measurements) as applied to RT systems are:

--inadaptability of most real-time operating systems (DOS RV M6000, OS RV M6000, DOS RV M7000, ASPO DOS M7000) for measurement, which makes it very difficult to use an intercept method;

--considerable expenditures of machine time when using the sample measurement method;

--poor accessibility of user to systems information on operation of the RT computer system.

Given below is an approach to measurement of a multiprocessor RT computer system operating in an automated process control system under control of a disk operating system based on an aggregated software system (ASPO DOS) that enables construction of measurement programs on the user level, considerable reduction of expenditures of machine time, improvement of reliability of on-line information about the load of the computer system.

The essence of the approach is that direct programmed measurements by the sample measurement method are made during "idling" of the computer system (intervals of natural time redunancy) that occurs for most RT computer systems in automated process control systems.

The transition to a measurement program used to determine the current status of the computer system and for processing of accumulated information occurs

at random times determined by the times of transition of each of the processors to a dynamic halt state; here despite the asynchronism of the processing operation, the measurement program ensures a strictly determined scale of time readings. An approximate estimate of the degree of loading of the computer system on interval Δt is given by the processor load factor

$$\mu = 1 - \sum_{j=1}^{n_0} \tau_j \theta \Delta t,$$

where n_0 is the number of redundancy intervals; τ_j is the magnitude of the j-th redundancy interval; θ is the number of processors.

Let us consider the algorithmic-program realization for the RT computer system based on the M7000 control computer operating in an automated process control system under control of a multiprocessor ASPO DOS that supports operation of a two-processor control computer complex with common field of direct-access storage and common network of I/O devices for two disciplines of processor distribution for the program divisions:

1) assignment of each of the processors to a certain division,

2) priority servicing of each of the program divisions by two processors.

The algorithm of the measurement program realizes three functional operations:

INT(1) - integration of "idling" time of the first processor on interval Δt with step T1;

INT(2) - integration of "idling" time of the second processor on interval Δt with step T1;

OBRA - processing and representation of data.

The structure of the first two functional operations with the use of ASPO DOS macrolanguage is shown below:

```
NAME INT(1),P,T1     NAME INT(2),P+1,T1
STIME BUS1,T1        STIME BUS2,T1
WAITE BUS1           WAITE BUS2
N:=N+1               N:=N+1
EXIT                 EXIT
```

Here P is the absolute priority of the first functional operation. Quite clearly in measurements of the load of the multiprocessor system the number of functional operations INT is equal to the number of working processors.

The third functional operation is realized by four sub-blocks: measurements, interpolation, statistical processing, formation of measurement protocols.

The measurement sub-block, using macro-operation RUN, interrogates program units INT(1) and INT(2), and computes the total "idling" time of the processors on interval Δl.

The interpolation sub-block calculates the instantaneous values of the load factors of the processors for a uniform time scale with step Δt and transmits the values to the display module. The procedure of interpolation for the i-th point is accomplished by the algorithm:

Step 1: The number of fixed points q is determined and the size of the intervals of loading of the processors is determined for the i-th point Δt_i:

$$q := E(l), \; \Delta t_i := (l - q)\,\Delta t,$$

where $l = (t_i - t_{i-1} + \Delta t_{i-1})/\Delta t$; t_{i-1}, t_i are the times of completion of processor loading for the i-th and (i - 1)-th points; Δt_{i-1} is the size of the loading intervals of processors for the (i + 1)-th point.

Step 2: Loading factors are calculated:

$$\begin{aligned}
&\mu_{l-q+1} := 1 - \tau_{l-q+1}/\theta\Delta t,\\
&\mu_{l-q+2} := 1,\\
&\cdots\cdots\\
&\mu_l := 1,
\end{aligned}$$

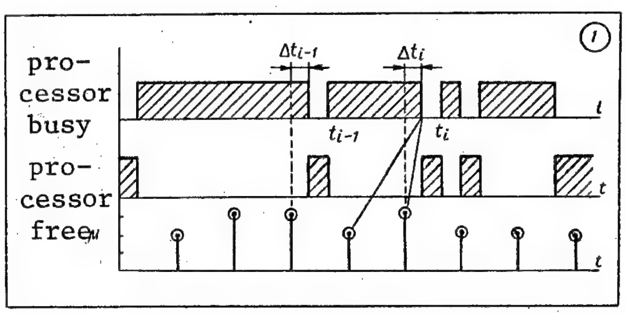
where τ_{t-q+1} is the overall interval of redundant time for the (i - q + 1)-th point.

Step 3: Operations

$$t_{l-1} := t_l, \; \Delta t_{l-1} := \Delta t_l$$

are performed.

The physical sense of the realized procedure is illustrated by Fig. 1.



The sub-block of statistical processing computes the mean value of μ_{cp} and the approximate value of the normalized autocorrelation function $R_{\mu\mu}$ for the sequence of instantaneous values $\mu(t_i)$:

$$(\mu_{cp})_{i+1} = \frac{1}{i+1}\,(\mu_{cp\,i} + \mu_{i+1});$$

$$R_{\mu\mu}(m\Delta t) = \frac{1}{\sigma_\mu^2\,(n-m)} \sum_{l=1}^{n-m} \mu^*(t_l)\,\mu^*(t_{l+m}).$$

where $\mu^*(t_i)$ is the centered value of the random quantity; $m\Delta t$ is the delay interval; σ_μ^2 is variance; n is the number of readout points.

The sub-block for forming measurement protocol accumulates the running values of μ in a buffer of capacity S, transcribes the buffer to a disk store, forms a graphic diagram of the processor load for the period of measurement, and outputs the protocal to a high-speed printer. In addition to the graphic load diagram, the measurement protocol contains information on the instantaneous load factors, the number of working processors, the average load and the measurement interval.

Technologically, the program module is formed as the three lowest-priority jobs on the user level that operate in one program division of the direct-access store with parameters P = 98, T1 = 500 ms, S = 100.

The structure and parameters of jobs INT(1) and INT(2) ensure simultaneous execution of these jobs in the artificial cycling mode only under condition that neither processor is loaded. One of the peculiarities of programmed realization of these jobs in the ASPO DOS is the necessity of blocking re-execution in case of a delay, and blocking output of operating system jobs to the dispatcher during execution of macro-operation STIME.

The program module is written in assembler, after composition takes up about 1.5K of the direct-access store, and uses intervals of the natural time redundancy ("idling" time) for operation.

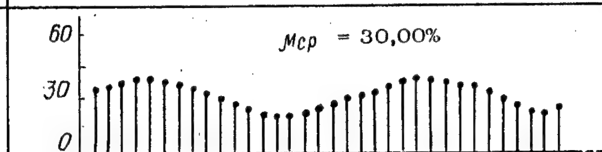
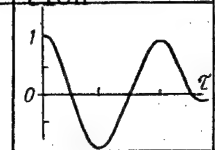
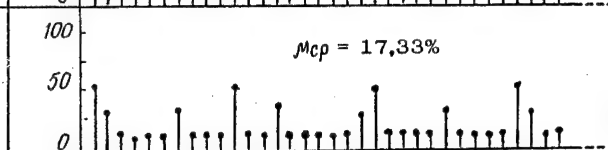
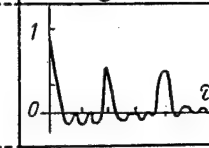
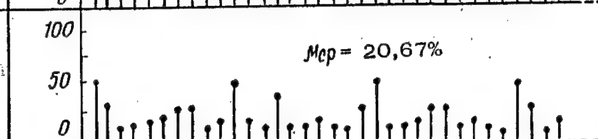
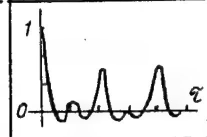
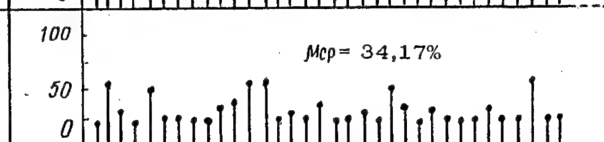
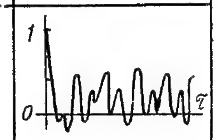
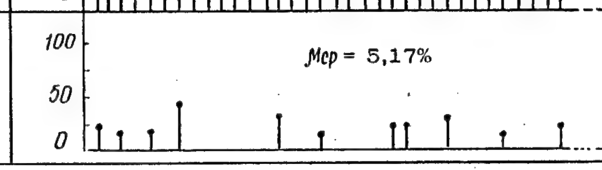
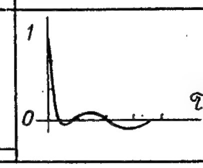
```
        PROCESSOR LOADING DIAGRAM (1)                          (2)
NN  LOAD   % 0                         50                        100
--------------I-------------------------I-------------------------I
01    28      I**************
02    52      I**************************
03    18      I*********
04    10      I*****
05    100     I**************************************************
06    18      I*********
07    66      I*********************************
08    76      I**************************************
09    16      I********
10    38      I*******************
11    26      I*************
       MEAN VALUE 41%
       MEASUREMENT INTERVAL:
          BEGINNING - 19 HOURS 24 MINUTES
          END       - 19 HOURS 35 MINUTES
```

Output of the entire measurement protocol, a fragment of which is shown on Fig. 2, is performed upon requirement by the operator and takes about 6 s.

One of the characteristic peculiarities of time diagrams of loading of RT systems in automated process control systems is the presence of stochastic and

deterministic periodic components: the stochastic component is due chiefly to the exogenic factor, while the deterministic component is due to cyclicity of the process of calculations in the automated process control system. By using the filtering property of the autocorrelation function, the deterministic component can be singled out and its parameters used as additional information for identifying the process with respect to load characteristics, e. g. for purposes of technical diagnosis. Fig. 3 shows loading diagrams and correlograms for different modes of operation of RT computing systems based on the M7000 process control computer complex.

3

Mode #	Type of programs	Loading diagram	Autocorrelation function (K_{MM})
1	Programmed harmonic waveform simulator		
2	Automated process control system: one DDC job; two control jobs		
3	Ditto in conjunction with background mode		
4	Automated process control system: two DDC jobs; four control jobs		
5	Systems processing programs		

Mode 1. Experimental verification of correctness of measurements on a periodic-load program generator.

Modes 2-4. Operation of actually existing automated process control system. In modes 2, 4, direct digital control (DDC) jobs are handled with supervision of material flows (control), while in mode 3, background work is done in a separate program division simultaneously with the automated process control jobs.

Mode 5. Development of new software components (translation, editing, composition, debugging).

In conclusion, let us note that the measurement program considered here is part of the peripheral software system of an automated process control system in a concentration mill of the Norilsk Mining and Metallurgical Combine, and

is currently in experimental operation. The results of measurements in developing and using the automated process control system hardware are being put to use for obtaining objective information on the degree of loading of the computer system, evaluating time characteristics of program modules, choosing an absolute priority system, spotting overloaded sections, planning the background working mode, evaluating efficiency of programmer use of machine time, and solving the following problems of functional diagnosis for the multifunctional ASPO DOS:

--choosing the structure of algorithms for verification test monitoring of hardware in the processor "idling" mode;

--on-line detection of malfunctions of the type of "cycling" and "hang-up" of user jobs by the method of tolerance control with respect to an average load factor;

--analysis of correctness of functioning of the computer system on the measurement interval with respect to parameters of the load diagram and autocorrelation function.

REFERENCES

1. Glushkov, V. M., Stogniy, A. A., Kushner, E. F., Pan'shin, B. I., UPRAVLYAYUSHCHIYE SISTEMY I MASHINY, No 2, 1980, pp 3-8.

2. Maksimey, I. V., "Funktsionirovaniye vychislitel'nykh sistem" [Operation of Computer Systems], Moscow, Sov radio, 1979, 272 pp.

3. Drummond, M., "Methods of Evaluation and Measurements of Discrete Computer Systems", Moscow, Mir, 1977, 382 pp.

6610
CSO: 1863/16

UDC 681.3.06

FEATURES OF PROGRAM INTEGRATION IN 'EL'BRUS' MULTICOMPUTER COMPLEX PROGRAMMING SYSTEM

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 (manuscript received 21 Jul 81, after correction 12 Apr 82) pp 92-96

[Article by Andrey Dmitriyevich Dobrov, engineer, ITM i VT AN SSSR [Institute of Precision Mechanics and Computer Technology, USSR Academy of Sciences], Moscow; Vladimir Mstislavovich Pentkovskiy, candidate of technical sciences, ITM i VT AN SSSR, Moscow; Marina Sergeyevna Prikazchikova, engineer, ITM i VT AN SSSR, Moscow; and Valentina Sergeyevna Chizhova, engineer, ITM i VT AN SSSR, Moscow]

[Text] Any programming system contains in its arsenal facilities for uniting programs from individual modules. The principle of modular programming and the practice associated with it of enlarging complexes by adding new modules make facilities for assembling programs an indispensable part of the majority of modern systems. Another aspect of assembly is the fact that it can enable the synthesis of modules of various programming languages. This makes it possible, first, to use ready programs in order to avoid the cost of repeated programming and of debugging algorithms and, second, to use languages with various functional capabilities in creating program complexes. Thus, the problem of the efficient synthesis of programs from units of various languages is fairly urgent from the viewpoint of practical programming.

In computers with traditional architecture, such as computers of the Unified System and BESM-6 computers, linkage editors and loaders perform the function of synthesizing modules. Compilers for individual languages generate object modules with a standard structure with a certain set of reference tables. Thus, in the programming system for YeS [Unified Series] computers the result of compilation is information of three types: text, and dictionaries of external characters and shifts [1]. This service information characterizes external entities to which there are references from a given module (unenabled external references) and names to which reference can be made from other modules (external names).

The linkage editing process in unification of modules consists in establishing a link between an external reference and a real entity and in forming a new text with the correction of address constants and the creation of new dictionaries. Language elements which can be matched by external names and references are program sections (assembler), procedures (ALGOL, PL/1) and subprograms (FORTRAN, PL/1). Unification of modules or editing of them are always regarded as operating

with units of a single lexicographic level which are replaced by data by means of parameters or via references to common regions.

Synthesis of a working program from modules of various languages is possible upon condition of the observance of agreements regarding linkages--the rules for using common registers and the method of transferring parameters and reserving the storage region. In uniting modules of high-level languages these agreements are performed by compilers. The inclusion of assembler modules necessitates directly taking into account agreements in programming.

In interlanguage synthesis of modules it is necessary also to match the operating environment in which modules of various languages can function. For example, for the successful operation of programs whose structure includes YeS operating system PL/1 procedures the existence of specific conditions is necessary any time control is transferred to this procedure (the so-called PL/1 environment). If for any reason the environment has not been preserved, e.g., in the case of transfer of control to the assembler program section, where it is disturbed, it must be restored by an explicit reference to standard procedures of the PL/1 library [2].

It must be emphasized that the presence of linkage editors and loaders in an operating system of the traditional type is fundamentally necessary. This is associated with the addressing systems used in the computer, as well as the strategy for static distribution of the memory and for loading of the program code and data into one and the same address space.

The linkage editor attaches to the loader module not only modules which have common (shared) on-line entities--variables, files, common regions--but also closed independent modules such as library interchange routines, standard functions and other independent subroutines. The majority of compilers require subsequent unification of the object module with library routines. And whereas unification of modules of the first category is necessary from the viewpoint of programming technology as a means of integration regardless of the components debugged, the attachment of closed units reflects the architectural features of systems associated with facilities for the interaction of modules.

The static union of program units creates a number of difficulties. First, enlarging the loader module can result in exceeding the size of the working storage allocated for it. It is possible to overcome this by inhibiting the attachment of those modules to which references are not made in a given run, and by means of planning an overlay structure. Secondly, modification of an independent routine requires repeating the step of editing all loader modules to which this routine is attached. Finally, let us mention that this procedure is inapplicable for programming the call of an independent module whose archive name is formed in the process of running a routine. In these cases it is necessary to use some kind of nonstandard mechanisms.

In the "El'brus" MVK [multicomputer complex] programming system another approach has been carried out to problems relating to interaction of programs. It makes it possible to a considerable extent to do away with the complexities arising in traditional systems.

Before proceeding to discuss the features of the "El'brus" MVK with regard to code loading, let us precisely define the significance of the term "program" in the context of this article. A program is a description of an algorithm in terms of the programming facilities. This description contains references to local entities, program parameters and archive entities, including files, other programs, reference guides, etc. Local entities of one program are not accessible to another. Interaction of programs is accomplished either by means of parameters explicitly transferred from one program to another or via the archive system. Internal entities of the requested program (files, common regions) never match local structures of the requesting program. For example, it is possible to form a program which represents a set of FORTRAN subprograms. Then the common regions of one set (if there are any) are regarded as entities localized in this program and these regions are not accessible to another FORTRAN program.

In the MVK instruction set the data address space is separate from the code address space. The latter has a fairly large capacity (2^{27} bytes). Conversion of a code relative address into a working storage address is performed with hardware support. As a result the program code possesses the property of shiftability and re-enterability [3].

Addressing of data in the MVK instruction set is only relative. Transformation of an address formed during compilation of a program text into a real address for the execution period is also hardware implemented.

These features make it possible for the operating system not to perform code correction in calling a program for execution. If attention is paid to the large size of the code virtual address space and the nature of hardware support for dynamic distribution of the working storage, then it is possible to formulate in the following manner the main aspects of program code processing.

A program ready to be run is stored in the form of a single code file in the secondary storage. This file is formed either as the result of complete translation of the text in the source language or as the result of static assembly of the code (integration) by means of the KOMPLEKSATOR [INTEGRATOR] program. In the process of running the working program, to the working storage are called (accessed) code segments of procedures which are being run at the time in question. With this no modification or correction of the procedure code takes place. Regions required for the data and code are reserved by the operating system dynamically in three sections of the memory with random location. Clearing of the working storage is also performed dynamically.

Storage in a single file of the code for closed procedures and of the subprograms mentioned above is irrational. Therefore, so-called dynamic acquaintance was used as the basic mechanism for interaction of programs not linked with one another through language external entities. The MVK operating system makes it possible to call a program found in another object code file. Acquaintance with such a program takes place only at the moment it is called. Subsequent references to it are possible without intermediate interaction with the operating system [3].

These features result in the fact that the presence of a linkage editor and loader of the traditional type is not fundamentally necessary in the "El'brus" MVK programming system. A set of facilities similar to linkage editors serves the purpose

of modifying existing code files and creating new ones. These operations are regarded in the system as additional capabilities, i.e., a type of system service. The KOMPLEKSATOR program which implements them is from the functional viewpoint a tool of the modular programming technology. It is important to note that a concept such as a loader module is lacking in the "El'brus" system. The program always exists only in the form of an object module and arrives in this form to be run, and the KOMPLEKSATOR or any similar system program is not used in accessing a code for execution.

Integration Tasks

The viewpoint discussed regarding the place of program integration facilities in the "El'brus" system has been a decisive factor in their development. The architectural features of the MVK and the capabilities of hardware and software have imposed new requirements on facilities for synthesizing program files. Of these it is possible to place the following under the heading of the most important.

The basic task of the KOMPLEKSATOR program is to add new procedures (subprograms) to the initial (basic) program and/or to replace existing procedures by their new versions. In both cases we will call the new procedures substitute procedures.

The majority of modern programming languages, including the "El'brus" autocode (simply autocode below), have a modular structure and developed program structuring mechanisms. Therefore, integration must accomplish the replacement and addition of procedures and subprograms described at any nesting level.

The necessity of using a finished program product in the form of program libraries and closed systems has introduced the requirement of interlanguage integration. In particular, inclusion in a program of fragments written in autocode functionally implements output to the basic programming language.

Natural development of the programming system implies the ability to include in integration new languages appearing in the system and this should take place at the lowest cost. It is necessary to take this fact into account both in creating the KOMPLEKSATOR program itself and in developing facilities on which developers of compilers for algorithmic languages rely.

Successful solution of the entire combination of problems listed above is based on the properties of the architecture of the "El'brus" MVK, as well as on the program decisions made in the system. Let us discuss its key features determining the main integration algorithms.

System Bases for Integration

The most important of the architectural properties of the MVK for solving integration problems is implementation in the software and operating system of the basic types of data and high-level language control structures.

Addition to the instruction set of tags (additional information describing the type of information in a machine word) makes it possible for the hardware dynamically to access operation execution algorithms as a function of types of operands.

This considerably unifies methods of accessing data (processing algorithms) in compilers for various algorithmic languages.

Hardware implementation of the procedure mechanism and introduction of the "procedure tag" data type unifies the representation in the computer of procedures and subprograms, the representation of a stack of generations of activations of procedures and their local data, methods of transferring actual parameters, methods of coding references to formal parameters, and the like.

For the purpose of implementing language types of data and structures, in the instruction set both simple types (integral, real and logical) and more complex, e.g., descriptors of files, are provided. Distribution of the memory in various compilers is accomplished by means of operating system primitives making possible implementation of files with constant and floating boundaries. On the whole the high level of primitives makes possible uniform implementation of semantically similar mechanisms in various programming languages and makes it possible to a greater extent to unify the representation of data, codes for accessing them and methods of memory organization. In addition to simplication of translation methods, this considerably facilitates development of integration algorithms, especially in the case of multilanguage variants.

A program object code file consists basically of the following components: procedure codes (code segments), a code segment reference guide and data on files of constants--invariable components of a program. In a certain sense the MVK code file is similar to an object module without reference guides for external linkages and shifts, but unlike a module (e.g., the YeS operating system) it does not have to be adjusted for a specific place in the RAM prior to running.

For components of the programming system requiring information on the text of the source program (e.g., dynamic diagnosis systems or the KOMPLEKSATOR program), each compiler generates additional information: data on all structural units of the source program and on correspondence of the code and text, as well as characteristics of language entities (variables, procedures, files) which are local for each structural unit.

The combination of code file and additional information for it has been given the name expanded object code file. It is in the form of a branched tree structure reflecting the logical structure of the program and containing complete information regarding the character and internal representation of program entities. The information structure was selected in such a manner to minimize the language-dependent part of the auxiliary information. The major amount is determined by requirements common to all languages (system requirements), which makes it possible to implement the majority of integration algorithms in a language-independent manner [4].

The addition to the code file is used not only in integration but also in the operation of other service programs. For example, it is necessary for the dynamic diagnosis system and for emergency output and for producing documentation regarding the program in terms of the source language. Minimization of the language-dependent part of the addition makes it possible to use these programs, too, as multilanguage facilities. In particular, thus is solved the problem of servicing programs consisting of fragments written in various languages.

Integration Problems and Methods of Solving Them

It is possible to single out three characteristic aspects of the integration process: formation of substitute procedures, correct distribution of resources in the expanded code file produced after integration, and also establishment of correspondence between external entities of the substitute procedure (simply the substitute, below) and entities of the source (basic) program.

Formation of the substitute can in principle be performed in a double manner. It is possible to employ the concept of an independently compilable procedure. In this case the programming language includes facilities for specifying the external entities which a given module can reference to or to which references from outside are possible. With this approach each compiler must make possible an independent compilation mode. These principles form the basis of development of the Burroughs computer programming system [5].

Another approach is used in the "El'brus" system. The substitute procedure is first described within a test program within whose structure it undergoes independent debugging. Then the KOMPLEKSATOR program separates it from this program and transfers it to the basic program, in which changes are made. At the same time a determination is made of the names to which references from this subprogram are possible. Entities of this type have been given the name external procedure names.

The structure of MVK instructions is such that analysis of the procedure code makes it possible to reveal uniquely the set of entities to which a module references in order then to correct references to them. This eliminates the need to use a dictionary of shifts (or reference guide similar to it).

The distribution of resources in the expanded code file produced after integration has been comprehended in some aspects. These include: proper and economical reutilization of reference guides (they represent a resource on file), accounting for redistributing and allocating the memory to new entities in the basic program (files of constants, code segments, variables), etc. A major requirement for integration facilities is the creation of a correct structure for the new code file and addition to it, in which real (new) values of attributes correspond to substitutes. Generally these operations do not depend on specific programming languages and are determined by the structure of the object code file.

Solution of the last integration problem consists in developing a general approach to problems of matching entities of the basic program and external names of the substitute. According to this approach the KOMPLEKSATOR program establishes correspondence between entities which are equivalent in the machine sense; they have an identical method of representation in the computer and similar algorithms for accessing them (access code). Here universality in implementation of a great number of types of data in various languages results in fairly extensive possibilities for linkage between entities and structures: procedures and subprograms; files and decision boxes (ALGOL) and files (autocode); a file (ALGOL) and general region (FORTRAN); and the like.

Relying on information in the addition to the code file, the KOMPLEKSATOR not only identifies external names and entities of the basic program, but also checks the correctness of links between them. Information on entities is concentrated in

dictionaries of local names of structural units of the source program. It is divided logically into two types:

Attributes relating to the internal representation of entities. These are the size of the memory allocated for it, the relative address (if there is one for it), constant value for invariable entities, etc.

Attributes which characterize entities from the viewpoint of the input language--static types, characteristics of files and common regions, specifications and methods of transferring procedure parameters and the like.

Let us discuss the question of the degree of correspondence of attributes of comparable entities of a module and the basic program. On one hand as the result of integration clearly correct results will be obtained if language attributes are identical in these entities. On the other hand, a complete check is far from always possible (or necessary).

Actually, programming languages which permit a certain dynamism can take part in integration. For example, the MVK autocode is characterized by full dynamics of types [4, 6]. Descriptions existing in a program are used basically for imparting local meaning to identifiers and for specifying the size (or format) of the memory region to be allocated to a variable. At the moment the program is run one and the same variable can take on values of a whole and real number, a procedure tag, file descriptor, etc. The language also has facilities for requesting procedures with an indefinite number of parameters of a previously unknown type.

For languages of the static class, such as ALGOL, all characteristics of entities are often unknown and the user receives a message regarding their improper use before the program is run. Therefore, in development of the KOMPLEKSATOR program the following principles were used as a basis.

If in the process of establishing correspondence between any two entities it was found that if only one of them is not characterized by a statically determined type, then only the simplest check is made. A check is made of the correspondence of attributes relating to the internal representation of entities--formats of elements and methods of referencing to them. This means, for example, that a simple variable, being an external variable for a procedure in autocode, can be linked with a common region or subprogram in FORTRAN, and it can also be linked with the name of an ALGOL file. This strategy makes it possible to interpret entities of one language via types of data of another more dynamic one. With the invalid use of an external name a situation can arise which will be a failure situation at the moment the program is run, which is regarded as the programmer's error.

And if entities are of the statically determined type, then the KOMPLEKSATOR program makes a diagnostic check to the full extent. It registers lack of correspondence of the number and specifications of formal parameters of procedures and subprograms, checks agreement of the dimensions of files, checks dimensions of common regions for FORTRAN, etc. If it is found that some common region is lacking in the basic program or an increase in its size is required, then a correction of the basic program's object code is made at the appropriate point. On the whole the possibility of such a check makes it possible to verify how accurately the programmer has adhered to the rules for interaction of program units of various languages

and helps to prevent a great number of error situations which cause interruption or unpredictable behavior of the program.

Conclusion

System implementation of the fundamental mechanisms of high-level languages in the "El'brus" MVK had a substantial influence on the structure and method of development of the programming system. This is true to an equal degree for creation of one of its components--the multilanguage program KOMPLEKSATOR. Summing what was said above, it is possible to single out the following characteristic aspects of integration in the "El'brus" system.

The status of program synthesis facilities has been changed in the programming system (as compared with the generally accepted). The KOMPLEKSATOR can be used as one of the tools of modular programming technology, but not as a system facility fundamentally necessary for preparing user programs.

Functionally, integration consists in fragmentary alteration of the object code file, i.e., makes it possible to replace procedures (subprograms) with new versions and to add new units. Here the code for remaining procedures is not subjected to any kind of editing.

Units extracted from a completely compiled and functionally closed program emerge, as a rule, as substitute procedures in the system, which makes it possible to transfer procedures from one program to another.

Modification of procedures can be performed at any nesting level.

The interaction of substitute procedures and the basic program by means of language (shared) entities is based not on program agreements, but on system (hardware and in the operating system) implementation of types of data and control structures of the most widespread high-level languages. This makes possible a high degree of unification of language mechanisms.

The sufficiently general structure of the universal interface for components of the programming system in the form of an expanded object code file makes it possible to implement in a system-oriented (language-independent) form the key integration mechanisms.

The last two facts are responsible for functioning of the KOMPLEKSATOR in multilanguage variants, permitting a diagnostic check of the correctness of the use of shared entities by the procedures of various algorithmic languages.

Bibliography

1. Lebedev, V.N. and Sokolov, A.P. "Vvedeniye v sistemu programmirovaniya OS YeS" [Introduction to YeS Operating System Programming System], Moscow, Statistika, 1978, 78 pages.

2. Malyshev, V.M. "Problems Relating to Interfacing Programs Written in Assembler Languages and PL-1 in the YeS Operating System," VOPROSY RADIO-ELEKTRONIKI. SER. EVT, No 3, 1979, pp 15-25.

3. Babayan, B.A. "Osnovnyye printsipy programmnogo obespecheniya MVK 'El'brus'" [Fundamental Principles of "El'brus" MVK Software], Moscow, 1977, 12 pages (ITM i VT AN SSSR Preprint No 5).

4. Volkonskiy, V.Yu. and Pentkovskiy, V.M. "Universal'nyy interfeys komponent sistemy programmirovaniya MVK 'El'brus'" [Universal Interface for Components of "El'brus" MVK Programming System], Moscow, 1980, 38 pages (ITM i VT AN SSSR Preprint No 28).

5. "Burroughs Corp. B6700 Program Binder (Form 5000045)," Detroit, Michigan, 1975, 66 pages.

6. Pentkovskiy, V.A. "Osnovnyye kharakteristiki Avtokoda MVK 'El'brus'" [Key Characteristics of "El'brus" MVK Autocode], Moscow, 1977, 28 pages (ITM i VT AN SSSR Preprint No 6).

8831
CSO: 1863/1

UDC 681.142.2

INKOL PROGRAMMING SYSTEM FOR PERFORMING COMPUTATIONS WITH INCOMPLETE INFORMATION

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 (manuscript received 21 Jul 81) pp 97-101

[Article by Genrikh Khaninovich Babich, candidate of technical sciences, Soyuztsvetmetavtomatika NPO [Scientific Production Association], Moscow; Irina Nikolayevna Falina, engineer, Soyuztsvetmetavtomatika NPO, Moscow; Leonid Fritsevich Shternberg, assistant, KAI [Kuybyshev Aviation Institute], Kuybyshev; and Tat'yana Igorevna Yuganova, junior scientific associate, Soyuztsvetmetavtomatika NPO, Moscow]

[Excerpts] In this article the results are discussed of the completed development of a programming system implementing a special-purpose algorithmic language oriented toward performance of computations with incomplete information. Incompleteness of information in the initial data means that in execution of a program certain data described in it, i.e., intended for participation in computations, have not been specified. Incompleteness of information in an algorithm means that in the program are contained references to functions the algorithm for computing which has not been specified.

The procedure for running a program under conditions of incompleteness of information calls for computation cycles and consists in the following. First computations are performed with the initial data whose values have been specified and references are made to those functions whose computation algorithm has been specified. As a result a solution is arrived at which is called a partial solution.

The partial solution contains variables whose value has not been specified, as well as references to functions whose description is lacking. It is possible that not all unassigned variables and not all references to undescribed functions are contained in the partial solution, since some of them could not be necessary for computations in the information context in question. The partial solution gotten in this manner can be analyzed by the user, as the result of which he will assign certain previously unknown values and specify part of the previously undescribed functions. Then a new computation cycle is performed and a partial solution is obtained corresponding to a more advanced stage of the computing process.

Such cycles of computation and analysis of partial results obtained are continued until the user eliminates the incompleteness of information or is satisfied with a partial solution obtained at a specific stage of the computing process.

The INKOL [INCOL] (from English INcremental COmputation Language--language for step-by-step computations) has the following major features.

The INKOL system is supplied to the user on magnetic tape for the M4030 computer in the form of a SYSLIB file containing S-, R- and C-libraries. The C-library contains the phases of the system ready for running and the S- and R-libraries make it possible in case of necessity to change certain parameters of the system, in particular, the capacity of the list storage. In addition, the existence of initial texts and R-modules makes it possible for the user to introduce into the system changes suggested by the developer for the purpose of modernizing it. For the same reason phases of the ALMO compiler are included in the C-library and modules of the ALMO interpreter in the R-library.

Bibliography

1. Babich, G.Kh., Shternberg, L.F. and Yuganova, T.I. "INKOL Algorithmic Language for Performing Computations with Incomplete Information," PROGRAMMIROVANIYE, No 4, 1976, pp 24-32.

2. Bogdanov, V.V., Yermakov, Ye.A. and Maklakov, A.V. "Programmirovaniye na yazyke ALMO" [Programming in ALMO], Moscow, Statistika, 1976, 118 pages.

8831
CSO: 1863/1

UDC 681.3.06;519.2:65

ARCHITECTURE OF LOCALLY DISTRIBUTED COMPUTER COMPLEX

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 (manuscript received 13 Feb 81, after correction 1 Apr 82) pp 64-66

[Article by Lev Davidovich Khatskevich, candidate of technical sciences, branch, VGPTI TsSU SSSR [All-Union State Planning and Technology Institute, USSR Central Statistical Administration], Voronezh; Nikolay Viktorovich Danil'chenko, engineer, branch, VGPTI TsSU SSSR, Voronezh; Mikhail Aronovich Leyzin, candidate of physical and mathematical sciences, branch, VGPTI TsSU SSSR, Voronezh; and Emil' Abramovich Shteynvil', candidate of physical and mathematical sciences, branch, VGPTI TsSU SSSR, Voronezh]

[Text] In this article a description is given of the architecture of a locally distributed computer complex created at the Voronezh branch of VGPTI TsU SSSR. The methodological basis for designing a locally distributed computer complex is the theory of homogeneous computing systems.* A locally distributed computer complex belongs to the class of homogeneous distributed computing systems. However, unlike homogeneous distributed computing systems in this complex the distances between computers are relatively short (up to 2 km). This makes it possible to implement relatively simple (92 integrated circuits) equipment for forming computers into a complex, to simplify algorithms for interchange between computers, and also practically to eliminate constraints on selection of the topology for links between computers.

The experience of creating applied systems based on a locally distributed computer complex has demonstrated that they will find an application in systems for controlling small enterprises, institutions and shops, as well as as a component of large integrated data processing systems.

Hardware

In developing the architecture of a locally distributed computer complex the key requirement is placing the computing capacity at the work place and offering the user the ability for direct communication with the computer. This is responsible

*Yevreinov, E.V. and Khoroshevskiy, V.G. "Odnorodnyye vychislitel'nyye sistemy" [Homogeneous Computing Systems], Novosibirsk, Nauka, 1978, 320 pages.

for the choice of a microcomputer as the basic computer. Since the computing capacity of a microcomputer is small, the necessity appeared of forming individual computers into a complex system by means of special equipment.

The "Elektronika-60" microcomputer was chosen as the basic computer for the locally distributed computer complex, which was due to the existence of a broad assortment of peripheral devices and the mass production of both the microcomputer itself and peripheral devices for it.

The minimal composition of the basic microcomputer is as follows: M2 processor board; P3 32K-byte-capacity RAM; PP1 512-byte-capacity ROM board; type IZOT-1370 disk storage control unit; UPO terminal connection interface; multicomputer organization unit; I7 board for controlling DZM-180 mosaic character printer; and a minidisplay.

The disk storage control unit, the multicomputer organization unit and the minidisplay were developed at VGPTI's Voronezh branch.

The multicomputer organization unit is connected to the "Elektronika-60" microcomputer's channel as a peripheral device. It is designed on the basis of circuits with a medium level of integration.

Key Characteristics of Multicomputer Organization Unit

Transfer method	Duplex or semiduplex
Interchange method	Synchronous
Interchange rate	28K bits/s
Interchange protocol	X.25 (first and second levels)
Method of connecting computers	Point to point

The IZOT-1370 disk storage control unit is executed with six printed circuit boards measuring 240 X 280 X 12.5 using microcircuits with a medium level of integration.

The characteristics of the disk storage control unit are similar to the characteristics of the SM-5402 unit.

The minidisplay, based on a home television set, consists of a unit for controlling output to the television screen and a sensor-type keyboard. The television set is connected to the control unit via its antenna input. The set of control characters is similar to that used in the "Videoton-240" display.

Key Characteristics of Minidisplay

Screen size	20 X 50 characters
Computer interface	Standard serial interchange board
Rate of interchange with computer	50 to 9600 bits/s

Structurally the computer is designed as a rack. All of the electronics are placed in a single structure. The electronics are powered by three series power supplies enclosed in a single structure.

System Software

The system software of the locally distributed computer complex includes an operating system (OS) and an instrument software complex.

Creation of the operating system for the locally distributed computer complex was performed in three stages: development of the operating system for a single computer; development of a 2-computer operating system; creation of a multicomputer operating system for the locally distributed computer complex.

Work on the first two stages has been completed and the corresponding MOS/60-1.0 tape operating system is described below.* In designing the operating system, and a multicomputer one in particular, one of the major problems is the problem of distribution of the system's resources. In the present system the least distributed resource is the computer. Thus, when it is necessary to solve a certain problem it is allocated the resources of several computers. This principle makes it possible simply to organize the efficient distribution of resources. However, its use imposes a limitation which is quite important from the traditional point of view--the impossibility of parallel solution of several problems on a single computer.

The trend in the development of computer technology consisting in making microcomputers and their peripheral devices less expensive and in making the cost of software higher, in our opinion, will soon result in widespread use of inexpensive reliable multicomputer systems based on microcomputers in place of expensive (by virtue of the complexity of the software and hardware) multiprogram computers.

In order to plan resources at the computer level information is needed on the participation of computers in the specific subsystem organized at the time of solving a problem. This information should either be found in each computer, and then control of the system will be of a distributed nature, or all the information will collect at a single center from which control of resources will be carried out. Hierarchical control of the system from a single computer seems intuitively simpler and more normal since it imitates the structure of teams of people. However, such a structure (even with the presence of a "substitute") is less reliable than a distributed structure.

A distributed system is more complex and results in circulation of large streams of information on the state of computers.

In this version of the MOS/60-1.0, which serves a total of two computers, the question of the choice of the principle for controlling resources is not so critical. Therefore, a simpler variant of hierarchical control has been chosen. The planning program fulfills the function of controlling the system's resources.

With the adoption of the principle of "indivisibility" of computers the possibility appeared of substantially simplifying the operating system by doing without working

*At the present time work has been completed on creation of the SPORK/60 16-computer disk operating system. It has been proposed to devote a separate article to a description of the SPORK/60.

with peripheral devices for interrupts. Actually this principle dispenses with multiprogramming and the probable area of application of interrupt facilities remains only the organization of input/output simultaneously with calculation (or of parallel input/output in several peripheral devices). However, an analysis of the usual requirements for input/output equipment imposed by applied programs demonstrates that the possibility of combining input/output with calculation arises extremely rarely. This is associated with the fact that as a rule input/output and calculation operations are combined in a single cycle and one operation depends on the results of another of these operations. In addition, usually input/output operations take up more than 90 percent of the time and therefore combining input/output with calculation (if this is possible) does not result in a great saving.

A very important property of the MOS/60-1.0 is its static and dynamic adjustability for the composition of the complex and the structure of connections between computers.

Static adjustment is performed in system generation. Here from ready modules of the operating system those are selected which are needed for working with a specific hardware configuration. Generatability of the MOS/60-1.0 is made possible by the modular structure of the system and is performed by means of the KOMPONOVSHCHIK-2V program, supplied with the standard software, as well as of a special program--the system generator.

Dynamic adjustment is performed directly before beginning to run a task and it includes possibilities for dynamic redistribution of several resources: devices, volumes and files.

In implementation of the MOS/60-1.0 the principle of layer-by-layer creation of software was employed: First the lower layer of the operating system, accomplishing input/output at the physical level, was placed on the "bare iron." With this a virtual computer was obtained possessing a greater set of possibilities than the original. Then by using the virtual computer produced a new layer of the operating system was created--the file control layer. Then the main dispatcher layer was built on top of the file virtual computer, etc.

With this organization of the work problems of organizing interaction between modules are drastically simplified, since each layer borders on only two others--the lower and upper. The procedure for introducing changes into the software is also simplified, since these changes can affect only layers going deep down from the directly corrected layer. With this changes are most often introduced only into a single layer.

This procedure, of course, results in certain overhead expenses in connection with the fact that if a service offered by a lower layer is required by one of the top layers, then this service must filter through all intermediate layers. However, the expenses are repaid with interest by the simplicity of the system's architecture and its great flexibility.

No small advantage of this method is the fact that it is possible to create applied programs at any level of the virtual computers. Here the requirement of succession is automatically observed, i.e., applied programs created in "low-capacity" virtual computers can operate also in subsequent versions of the operating system.

System fields and tables are the basic means of communication between operating system modules, and they are the most changeable part of the operating system. It is feasible to classify data according to membership in various layers of the operating system: general-system tables; data belonging to a layer; and data which are used by all layers above the one in question. On this basis it is possible to organize access to them.

The system developed was used for creating a system for controlling the production operations of one of the departments of the "Elektronika" PTO [Maintenance Station], a system for analyzing the production and economic operations of oblast enterprises in the Voronezh CPSU obkom, and a system for managing an urban economy.

8831
CSO: 1863/1

UDC 681.324.00(045)

PAROM HOMOGENEOUS MICROCOMPUTER COMPUTING SYSTEM

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 (manuscript received 11 Feb 81, after correction 20 Mar 82) pp 66-70

[Article by Vasiliy Grigor'yevich Gorin, engineer, branch, VGPTI TsU SSSR [All-Union State Planning and Technology Institute, USSR Central Statistical Administration], Smolensk; Mikhail Yakovlevich Indenbaum, engineer, branch, VGPTI TsSU SSSR, Smolensk; Vladimir Mikhaylovich Korolev, engineer, branch, VGPTI TsSU SSSR, Smolensk; Valeriy Vladimirovich Parokhodov, engineer, branch, VGPTI TsSU SSSR, Smolensk; Grigoriy Yesaulovich Pozdnyak, candidate of technical sciences, VGPTI TsSU SSSR, Moscow; and Mikhail Arkad'yevich Khvostantsev, candidate of technical sciences, branch, VGPTI TsSU SSSR, Smolensk]

[Text] Homogeneous microcomputer computing systems (MOVS's) substantially expand the range of application of microprocessor equipment and improve its economic efficiency, making it possible to solve not only very simple problems for which the capacity of a single microprocessor is sufficient, but also complicated problems which can turn out to be not under the power of even large computers. The methodological foundation for designing such systems is the model of a team of computers [1, 2], based on the principles of parallel organization, variability of the structure and structural homogeneity.

The use of series-produced microcomputers as elementary machines (EM's) makes it possible with minimum modification of equipment to create modular add-on-type computing facilities. On the other hand the limited nature of the resources of microcomputers, their rigid structure and the primitive nature of their architecture present a number of complex system-use problems. These include: the need to achieve throughput of the system proportional to the number of elementary machines; ensuring reliable operation; and development of efficient software.

Unlike homogeneous computing systems consisting of large computers and minicomputers, MOVS's must contain a far greater number of elementary machines to achieve the prescribed throughput. This makes it necessary to develop special system facilities which in combination will solve problems of reliability and efficiency, which do not depend on the number of elementary machines in the MOVS, and permitting implementation in the form of large-scale integrated circuits.

In solving a number of complicated economic reporting and data processing problems MOVS's must, in addition to high throughput of hardware, also ensure a high level of operating system service functions. The primitiveness of the architecture of

microprocessors and the limited nature of their working storage have made it necessary to find a new approach to developing the operating system and applied software.

At the present time any large computing system project must take into account the rapid rate of improvement of microprocessor facilities, which have made it possible gradually to implement an ever greater number of operating system functions through hardware.

The PAROM (Parallel'naya Raspredelennaya Obrabotka na Mikromashinakh [Parallel Distributed Processing Using Microcomputers]) homogeneous microcomputer computing system uses as elementary machines microcomputers of the "Elektronika-60" series, which are program compatible with "Elektronika 100-25" and SM-4 minicomputers. This makes it possible to use as the starting software for the MOVS the FODOS, RAFOS and OS RV [real time operating system] operating systems. These operating systems can perform both the role of basic operating systems for the MOVS and the role of instrumental systems for developing new operating systems which do not represent an expansion or superstructure of basic systems.

Logical Structure of System

The basis of the logical structure of the MOVS is the add-on system processor (fig 1). The development of system facilities for both a single special-purpose and variable processor was caused by the need to solve three interrelated problems. The most important of these is the creation of hardware support for a functionally complete system operations basis. Basis operations must be performed with prescribed speed regardless of the number of computers in the system.

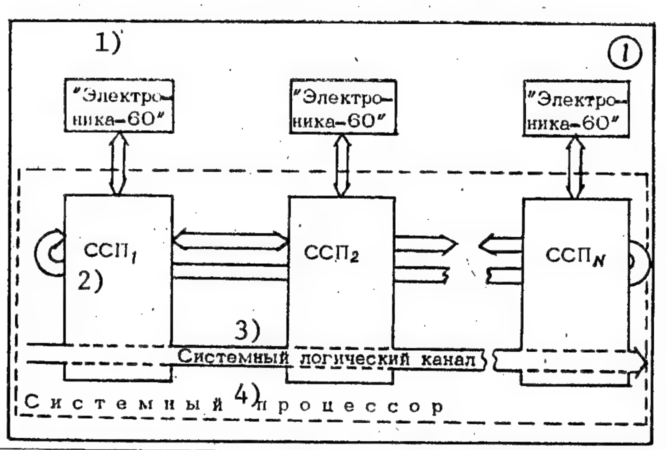


Figure 1. Logical Structure of PAROM System

Key:

1. "Elektronika-60"
2. SSP_1 [system processor section]
3. System logic channel
4. System processor

The second problem is initialization of the system, which can be accomplished either in the mode of communication with a console terminal or by means of a program or

microprogram stored in a power-independent memory. The absence in the majority of elementary machines of a unit performing the functions of a console terminal, as well as of a power-independent memory, complicates solution of this problem.

The third problem is to enable comprehensive self-diagnosability of the system, necessary for implementing survivability conditions and automating use of the system. The complexity of solving this problem is due to the relatively low reliability of elementary machines, the great number of machines in the system, and the absence of standard facilities for self-diagnosis in "Elektronika-60" microcomputers. In addition to the problem of diagnosis, of important value is the problem of structural reliability, associated with it. It consists in creating a structure for the system processor which will not permit the influence of failures of some elements on the normal functioning of others.

An analysis of these problems made it possible to single out a unique basis for operations implemented by the system processor in interaction with elementary machine processors, which is used jointly to enable specific modes of functioning of the operating system. The system processor is in the form of a set, connected in a regular manner, of identical system processor sections (SSP's), each of which interacts with its own elementary machine and neighboring sections in performing its part of a system operation. Each SSP performs three key functions: simulates a console terminal in initialization and functioning of the system; implements the system operations basis; identifies failures of elementary machines and informs the system of them.

The structure of an EM and SSP is shown in fig 2. A parallel high-speed interface with the computer's channel makes possible interchange through interrupt and program interchange of the SSP with the elementary machines. For initializing the system, consisting in loading the operating system into all elementary machines, a special floating console terminal mode is implemented. This mode makes it possible to load the operating system into elementary machines not having a console terminal, by using the console terminal of a single elementary machine. The floating console terminal is a shared system facility which is used also for diagnosing the system and debugging programs.

The operating unit of the SSP interacts with the elementary machine's processor via a parallel interface and with other EM's of the system via the system channel (system common logic line). The operating unit makes possible performance of the following operations: setup, interchange, synchronization, and generalized conditional and generalized unconditional transfers.

Setup of any elementary machine consists in transferring to it a control character indicating whether it is participating in execution of a system statement or not. The setup variant which has been implemented is designed for a system consisting of no more than 104 machines. Setup is performed in parallel group by group. Thirteen elementary machines of a single group are set up in a single operation; therefore, a total of eight operations is required for setting up a system of 104 elementary machines.

Interchange is divided into two elementary operations: transfer and reception. The first reduces to transfer of data from the memory of an elementary machine to the system channel, and the second to input of information from the system channel

into the machine's memory. At any moment of time there can be in the system only one transferring EM and as many receiving EM's as desired. Such an interchange system is called translational. Pipeline, duplex and other interchange systems can be implemented on its basis by means of the setup. EM's go into interchange at various moments of time, but this operation does not begin until all machines of the system taking part in it have performed synchronization.

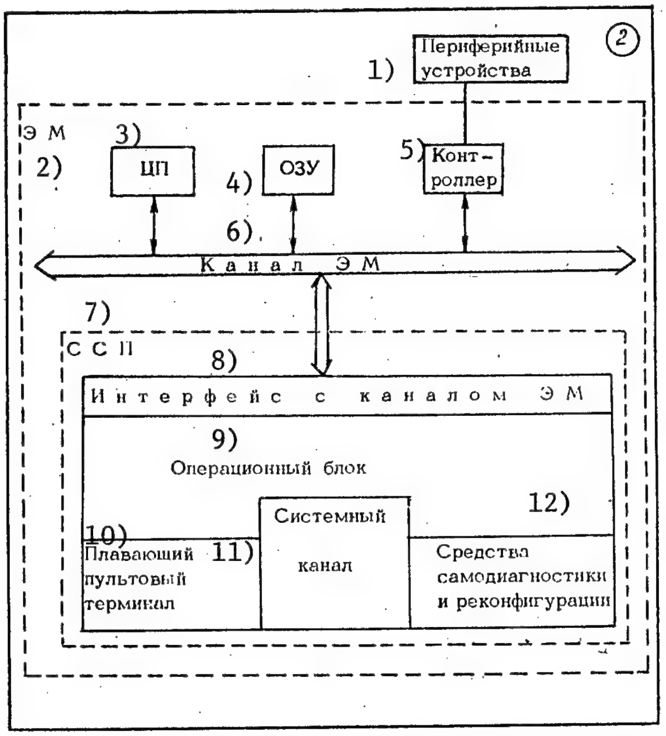


Figure 2. Structure of Elementary Machine

Key:

1. Interfaces
2. Elementary machine
3. Central processor
4. RAM
5. Controller
6. Elementary machine channel
7. System processor section
8. Interface with elementary machine channel
9. Operating unit
10. Floating console terminal
11. System channel
12. Self-diagnosis and reconfiguration facilities

A generalized conditional transfer consists in simultaneous program branching in all set-up machines of the system depending on the value of the system level. The system branching condition takes on a value equal to the value of some logic function from variables specified in each set-up EM.

A generalized unconditional transfer consists in interruption of all set-up EM's and reception of an interrupt code from the system channel. This operation makes it possible, in addition to registering the setup control character, to register a special interrupt mask program-set in the SSP state register. This makes it possible for EM's to inhibit propagation in them of the effects of this operation. With

asynchronous interaction of EM's the system channel is the critical resource of the system.

Special facilities, the method of implementing which is shown in fig 3, have been added for the purpose of controlling the system channel. The series-connected flip-flops of all SSP's form a distributed shift register through which a unit impulse travels at a frequency determined by the pulse generator. The flip-flop of only one SSP can be set at each moment of time. If none of the EM's requires the system channel, then the unit impulse is moved without delay through the distributed shift register (fig 4a).

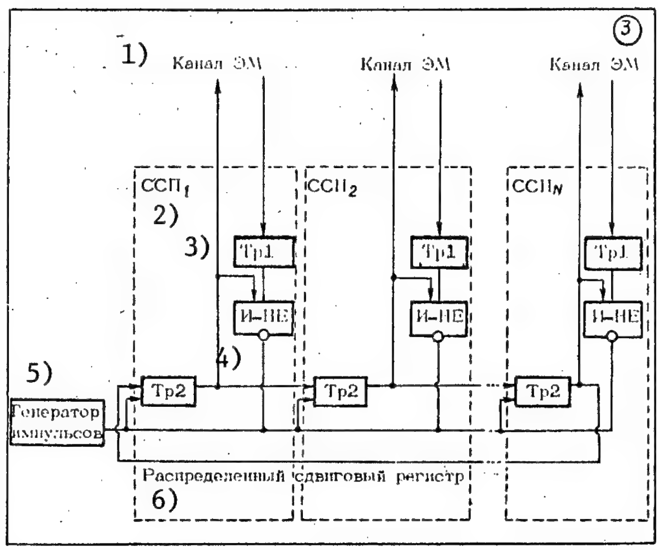


Figure 3. Facilities for Distributed Control of System Channel

Key:

1. EM channel
2. SSP
3. Flip-flop No 1
4. NAND gate
5. Pulse generator
6. Distributed shift register

Before working with the system channel, the EM sets the channel request flip-flop. With the arrival of a unit impulse in the flip-flop of a given SSP, a zero level appears in the output of the NAND gate, which halts further movement of the pulse through the shift register.

Having identified the presence of a channel grant signal from the flip-flop's output, the EM can perform operations with the system channel. After termination of the operation the EM clears the channel request flip-flop at the same time permitting operation of the pulse generator. The unit impulse continues to move through the shift register. If two or more EM's wish simultaneously to occupy the system channel, then it will be granted to them in turn, as determined by the moving impulse. In fig 4b it is shown how distribution of the channel occurs in the case of a simultaneous request from EM1 and EM2.

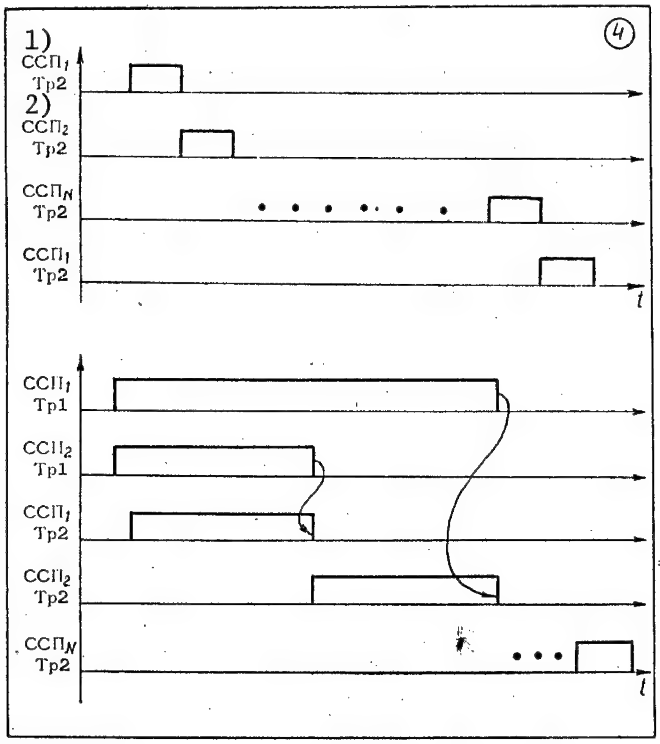


Figure 4. Operation of Facilities for Distributed Control of System Channel: a--system channel free; b--system channel occupied

Key:
1. SSP
2. Flip-flop No 2

Thus, the facilities introduced make possible use of the channel by only one or by several EM's desiring to occupy it. No EM will stand in line for an unlimitedly long time.

Facilities for self-diagnosis and reconfiguration of SSP's enable identification of the majority of failures and output a generalized interrupt signal to all computers of the system. Facilities for ensuring survivability of the system make it possible to determine malfunctioning EM's, to perform reconfiguration and to continue normal functioning of the system with a smaller number of EM's. The system identifies failures within an EM usually associated with a channel accessing error, failures of the EM's power supply and synchronization circuits, errors in transfer of information through the system channel, and also failures in reconfiguration circuits.

Any of the failures mentioned will result in a loss of an EM for the system. However, these failures are not spread through the system since EM's are connected to the passive channel in parallel and access to the resources of any machine is accomplished exclusively by program.

Special power supply redundancy facilities are provided in the system making it possible to maintain its serviceability with a loss of power in an EM. For the kinds of failures listed the PAROM system possesses the property of maintaining its serviceability right down to a single EM.

The measures used for ensuring the system's failure immunity still do not solve the problem of reliability as a whole. Hardware for counteracting errors is lacking in the system. Computations immune to errors can be realized by the program method, using traditional methods. The system is not immune to intentional and random program errors, which can result in damage to the operating system, which is the consequence of the absence of hardware for protecting the memory in EM's.

Unlike a system with a hierarchical structure of switches and connections between computers, e.g., the St* [3], each EM of the PAROM system does not directly address the total field of the working storage. Access to common data is accomplished by employing high-efficiency group interchange operations and hardware synchronization of branches on account of the structure of a special class of parallel programs oriented toward the architecture of homogeneous computing systems [4]. The time for access of any EM to the memory of other machines does not depend on their position in the system.

One of the main criteria for choice of the structure of the PAROM system was achievement of serviceability acceptable for an industrial system with an average number of EM's in the system equal to 20. The structure of the common passive line to which the EM's are connected in parallel ensures high hardware reliability. SSP's connecting computers to the system channel are identical for all EM's. Each SSP is executed with a single circuit board containing 80 microcircuits, which is four times less than the hardware input of the St* system.

System Peripheral Devices

The PAROM system makes it possible to work with a broad set of peripheral devices. Furthermore, generally their uniform distribution over the system's EM's is not required. In the minimum configuration it is sufficient to have a display terminal and a magnetic disk storage in only one of the EM's through which initialization of the MOVS is accomplished. Input of information into the system can be performed from the keyboard of a display, punched tape, from magnetic media, as well as from non-standard user devices (sensors, measuring instruments, etc.). Peripheral devices can operate both via a serial (at a distance up to 1.5 km) and through a parallel interface.

A display, printers, punched tape, magnetic disks and tape, as well as non-standard user devices, are used as output units. SM-5400 and SM-5401 magnetic disks, "Elektronika NGMD-70" floppy disks and IZOT-5003 magnetic tape are used as external memory devices. The system is compatible with YeS [Unified Series] computers with respect to magnetic tape data formats.

One of the key system components is the SI-T serial 2-port interface based on a K580IK51 microprocessor large-scale integrated circuit. It makes it possible to accomplish data interchange at a distance up to 1.5 km through a physical circuit without modems at a rate of up to 9600 bits/s.

Another important device in the PAROM system is the "Elektronika-60" interface adapter for an "Elektronika 100-25" and SM-4 minicomputer interface, which makes it possible to use them as EM's in quasi-homogeneous systems. All peripheral devices are connected to the channels of a system's EM's and their structure is determined according to specific applications.

Architecture of System

The PAROM is a multilevel system (fig 5). The MOVS's hardware level includes in its structure the instruction set of the individual microcomputer contained in the MOVS and a basic set of system operations for interaction between computers.

On top of the hardware level is constructed the operating system, which in turn is 2-leveled. The lower level of the operating system (the base level) is a natural extension of the software and in the future should be hardware implemented to an ever greater extent. The basic operating system makes possible an interface between the problem-oriented operating system and the hardware level, offering the problem-oriented operating system a general-purpose set of basic facilities for controlling the MOVS and serves as a foundation for creating a wide range of problem-oriented operating systems.

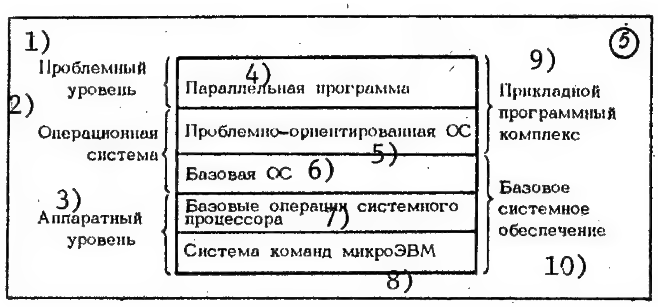


Figure 5. Architecture of PAROM MOVS

Key:

1. Problem level
2. Operating system
3. Hardware level
4. Parallel program
5. Problem-oriented operating system
6. Basic operating system
7. Basic operations of system processor
8. Microcomputer instruction set
9. Applied program complex
10. Basic system software

For each of class of problems to be solved in the MOVS its own upper level of the operating system is created--a problem-oriented operating system which offers the problem programmer a set of efficient facilities specially oriented to solving a specific class of problems and which makes possible the required modes for functioning of the MOVS. The assignment of two levels in the operating system makes it possible, in spite of the limited resources of individual microcomputers included in the MOVS, to create a large number of efficient operating systems making possible the solution in the MOVS of various problems.

At the problem level parallel problem-oriented programs are created for solving one or more classes of complicated problems. Creation of the software for the MOVS is itself a complicated problem. For solving this problem a special problem-oriented MOVS is being created which includes in its structure high-level languages with the ability for explicit representation of parallelism or automatic multiprogramming, facilities for debugging parallel programs and facilities for automating the technological process of creating software.

Thus, two types of systems are distinguished: a process, designed for creating software, and an executing, designed for using already prepared software. Distinguishing between two types of systems makes it possible to create simpler and more efficient systems.

Packages of applied programs used in the MOVS must consist of four parts. The bottom part of a package is the level of the problem-oriented operating system; here a medium is formed for operation of the upper part of the package, where problem tasks are solved (the problem level). The right half is in the process complex and contains programming facilities, and the left half is in the executing complex and enables functioning of the package when solving problem tasks.

Application of System

The high throughput and add-on capability of the PAROM MOVS, the ability for joint use of the system's resources by various processes, and flexible interaction between computers make it possible to construct various applied systems on the basis of the MOVS.

In institutional systems which automate the processes of processing documents and checking fulfillment, planning and control, in the MOVS a subsystem controlling the institution's data base and a subset of EM's servicing users are set aside. Terminals (displays) with printers for making paper copies from the screen are installed directly at work places. Magnetic disks and tape, as well as other key equipment of the system, are concentrated at a single place and are attended to by the system operator.

For effective functioning the disk storage must have considerable capacity and be unremovable with serial operation of the MOVS in the real-time mode. The PAROM system, possessing great computing resources (the throughput with 20 EM's can reach 2 million operations per second), can be used for solving complicated computing problems, for processing large amounts of information in real time, as well as for simulating and designing other systems for parallel data processing.

Another important application of the MOVS is as an ASUTP [automated system for controlling a technological process]. Here the EM's serve individual groups of sensors and actuators which accomplish local control. A subsystem of several EM's can be formed when necessary for processing a large stream of data. On account of its capacities for efficient data interchange the system can make a master decision, e.g., in an emergency situation. The ability to change the system's structure and the presence of various kinds of redundancy make it possible to guarantee high reliability and survivability indicators for all applications.

Bibliography

1. Yevreinov, E.V. and Kosarev, Yu.G. "Odnorodnyye universal'nyye vychislitel'nyye sistemy vysokoy proizvoditel'nosti" [High-Throughput Homogeneous General-Purpose Computer Systems], Novosibirsk, SO Nauka, 1966, 308 pages.

2. Yevreinov, E.V. and Khoroshevskiy, V.G. "Odnorodnyye vychislitel'nyye sistemy" [Homogeneous Computing Systems], Novosibirsk, SO Nauka, 1978, 318 pages.

3. Fuller, S.Kh., Usterkhut, D.K., Raskin, L.V. et al. "Multiprocessor Systems: Review and Example of Practical Implementation," TIIER, Vol 66, No 2, 1978, pp 135-150.

4. Mirenkov, N.N. "MINIMAKS--Collective-Use Computing System," VYCHISLITEL'NYYE SISTEMY, No 60, 1974, pp 115-128.

8831
CSO: 1863/1

INEFFICIENCY IN SOVIET COMPUTER USE

Moscow MOSKOVSKAYA PRAVDA in Russian 7 Sep 82 p 1

[Excerpt] Computer technology is being more and more extensively introduced into practice in various sectors of modern industry. Powerful computers are being used as the basis for setting up automated systems to control technological processes, departments, enterprises and whole sectors of the national economy. The use of computers is also having a considerable effect in scientific institutions, planning institutes, and design offices. Automated design systems have come into wide use of late, appreciably helping to accelerate the development of new equipment and technological complexes, and reducing expenditures for producing these facilities. For example, in the design of induction motors at the All-Union Scientific Research Institute of Electromechanics the use of an automated design system has accelerated the output of technical documentation five-fold, improved optimality of plans, and on this basis has saved nearly a thousand metric tons of electrical engineering materials, and 150 million kilowatt-hours of electric energy per year.

Electronics has become a reliable helper for scientists and designers in some institutes of the USSR Academy of Sciences, as well as in such sector-wide scientific institutions as the All-Union Scientific Research Institute of Construction and Road Machine Building, the All-Union Scientific Research Institute of Planning of Hydroprojects imeni S. Ya. Zhuk, and the All-Union Scientific Research and Planning Design Institute of Metallurgical Machine Building. Introduction of research automation systems has improved work efficiency of specialists by a factor of 3-5.

Obviously there is no need to prove the effectiveness of extensive introduction of automated systems based on modern computers in all areas of scientific research today. And there are adequate capabilities for carrying out this task. There are about 850 computer centers and subdivisions of separate organizations equipped with computers operating in Moscow right now, the inventory of modern third-generation computers numbers in the hundreds (the number has increased by 187 units in the last three years alone). And a very appreciable number of the new computers have gone to work for Moscow scientists and designers.

But the installation of a new computer in the computer center of the institute and assignment of its work are only the beginning of introduction of this

powerful and (lest we forget) expensive piece of equipment. Daily and persistent effort is required to get full value and high efficiency out of using the inventory of electronic equipment in the interests of improving economy of developments.

Moscow science workers as well as specialists in many areas of industry still have plenty of unused reserves in this direction. The actual useful work time for computers is about 80% of the calendar time, i. e. out of every hour about 12 minutes is wasted when the computer could have been put to good use. Occasionally a lot of time is lost due to idle periods, recalculations, and computer down time for preventive maintenance and repair. For example the average daily work load for computers in some organizations of light industry and foods amounts to barely more than nine hours.

Such figures are far from optimum. However, even these levels still remain an unconquered peak for many institutions. And it is often science collectives that are outstanding in their irrational use of computer equipment, even though their work is totally suitable for the very class of automation capabilities that has been defined for present-day electronics. For example, a check has shown that the excellent YeS-1022 computer at the All-Union Scientific Research Institute of Medical Instrument Making has had a work load of only about four hours a day, and [the same computer] in the All-Union Scientific Research Institute of the Cable Industry--five hours. Similar figures could be quoted for some other organizations as well. It wouldn't hurt them to emulate the experience of really efficient use of computers in the science institutions of such sectors as communications, construction and certain others.

Inadequacies in the use of computers can be attributed to a number of causes, many of which are a consequence of a spendthrift and lackadaisical attitude toward the equipment on the part of specialists of scientific research institutes and design offices. For example, it has been found that some computers are not used because of delay in making them operational, and failure to meet deadlines specified for assimilation. Many computers stand idle for a totally invalid reason--for lack of so-called computer work and personnel. There are often cases where the workers in science institutions, knowing of the impending arrival of computer equipment, cannot find time to get the required rooms ready. Expensive equipment stands idle for months, awaiting "quarters". The losses amount to hundreds and thousands of rubles. Such cases have been noted in organizations of the Ministry of of Chemical Machinery, the USSR Ministry of Higher Education and Gosstroy SSSR.

Many failures in the use of computer equipment are related to poor consideration given to hardware both in the computer centers themselves and at customer terminals. There is not enough software for automating research, or technological complexes for automating programming. Here there is food for thought for the directors of Gossnab SSSR, the Instrument Making Ministry, and other ministries and agencies responsible for providing all necessary facilities for computer complexes. Applied software is poorly organized -- according to the USSR State Commission on Science and Technology, users are inadequately provided with necessary facilities. Data processing technology is incomplete, and in this connection some of the workers in the computer center are occupied with nonproductive keyboarding and keypunch work.

The situation is aggravated further by the fact that many agencies and organizations (including scientific institutions) are making efforts to develop their own computer centers, and then are unable to keep them completely loaded. Eighty-five percent of the Moscow computer centers are local, serving only one enterprise or organization. Moreover, such practice is also occurring in the current five-year plan: forty centers of this kind are to be set up.

Each of the enumerated facts has specific and totally eradicable causes. Directors, communists, all workers in scientific research institutes, design offices, and organizations that are in one way or another associated with using computer equipment must radically re-examine their attitude toward this important matter. Maximum loading of all computers and other electronic equipment is not a senseless wish, but a necessity dictated both by the general course toward intensification of work and by the nature of today's research and development. The initiative of each specialist, each director of a scientific subdivision, is important here.

Another considerable reserve is solution of a problem that faces many computer centers: organization of work, optimum placement of personnel. The situation existing in certain organizations of the Ministry of the Electrical Engineering Industry and some other sectors must not be taken as normal: only half the workers in computer centers, departments and laboratories that have computers are directly occupied with programming and use of computers. At the same time, management machinery here is excessively large, comprising nearly a sixth of all personnel. Meanwhile, the computer centers of these organizations are experiencing a shortage of skilled specialists. Computers are being operated on a single-shift basis. And all the while there is a need and real possibility for two-shift if not three-shift work loading of electronics.

6610
CSO: 1863/9

TWENTY-FIFTH YEAR FOR VILNIUS COMPUTER PLANT

Vilnius SOVETSKAYA LITVA in Russian 21 Sep 82 p 2

[Excerpts from article by G. Litvintsev: "The Whole Nation Can be Seen"]

[Excerpts] The output of Sigma Production Association of Vilnius is the result of collective labor of hundreds of enterprises and scientific design organizations of the nation.

M5100 computer complexes and YeS 7010 keypunch equipment are the principal products of Schetmash.

The Schetmash Plant of Vilnius is comparatively young; it has its twenty-fifth birthday this year. Actually, this is the age of the entire sector. It was in the fifties that electronics began really developing in our nation. And now it is one of the largest sectors of the national economy. There are electronics enterprises in more than seventy krays and oblasts.

By prohibiting the sale of technology and special technological equipment in the USSR, especially for microelectronics, the United States and other capitalist nations have sought to interfere with the development of this sector in our nation. But their attempts have been futile. The USSR has developed its own electronic machine building, and has carried out complete retooling of all enterprises and institutes.

A characteristic trait. At the beginning of the Tenth Five-Year Plan, scientists in the United States said that the Soviet Union was eight or ten years beheind in microelectronics. But after studying some models of our circuitry in 1979 they estimated this lag at two or three years. And in the January issue of the U. S. journal ELECTRONICS for 1981, it is acknowledged that the "technological base and skill of specialists are enabling the Soviet Union to make integrated circuits of the same quality as in the United States." In the synopsis, the editorial staff suggests that the USSR has even better integrated circuits of the highest technical level.

During the Eleventh Five-Year Plan, our nation is keeping up the high pace of development of the electronics industry. The field of application of electronic equipment is continually expanding, and the number of consumer enterprises is increasing. Next comes extensive introduction of electronics in

motor vehicles and agricultural machines, in railway transport. To do this, during the Eleventh Five-Year Plan we will make millions of microprocessors and tens of thousands of micro- and minicomputers, having a revolutionizing effect on many sectors of the national economy.

A leading role in production of new computer equipment has been given to the Sigma Association, the nation's principal manufacturer of small computers of the M500 family. Demand for these computers in the national economy is continually increasing, as are requirements for machine "capabilities." In 1983 the association will begin production of the new series of SM1600 small computers. These will utilize the latest advances in computer technology.

Taking part in the development and production of new computers are many scientific and production collectives. For example, some designs have been developed by engineers of the Impul's Scientific Production Association of Severodonetsk, and the Elektronmash Association of Kiev. The Kievans are making a printer as well. Video terminal devices are being made by Terminal Production Association of Vinnitsa. The Penza Affiliate of the Scientific Research Institute of the Instrument Making Industry has helped to improve the coating process. Workers at Leningrad Scientific Research Institute of Abrasives and Grinding have produced special deep-grinding wheels for the plant that can give high surface quality. There are many examples of this kind of cooperation.

Early this year, Sigma Production Association incorporated the Soyuzelektronmash Industrial Association into its organization.

"This is a natural step" in the opinion of A. Nemeykshis, chief of the special design office of computational machines, and honored engineer of the Lithuanian SSR. "It will help to accelerate the process of development and introduction of new computer systems. More efficacous solutions are being found for problems of integrating production on a nationwide scale, planning and software."

Even now, within the framework of nationwide integration, Sigma is making its contribution to developing production of all models of computer technology. Output of series SM5408 magnetic disk storage units has been recently organized at the enterprise.

The entire collective has prepared for production of these storage devices. But perhaps those who have been most intensely involved are workers in alignment. They have gone through special courses organized in the special design office of the association. The first series-produced storage unit was adjusted by representatives of three teams-- those of S. Kats, Ya. Vebra and the team named in honor of the Twenty-Sixth CPSU Congress and led by V. Simanavichyus. At the end of the shift of 30 June 1982 the adjusters reported: the first item of the most complicated product for the enterprise has been made! Still remaining are State tests; some technical and organizational problems are still unsolved. Yet a great new step has been taken in development of the enterprise.

Where will the new goods made by Sigma be going? The storage units will be used by the Elektronmash Production Association of Kiev and the Impul's Assocition of Severodonetsk for general-purpose computers.

To what addresses in general will goods be sent from the Vilnius association where they are made?

N. Grishchenko, chief of the sales department of the association:

"Computers with the Sigma trademark can be found in nearly all cities of the Soviet Union, including regional centers. They go into the makeup of a variety of computer systems, primarily the "Minsk" computer complex. They are extensively used in the system of the Central Statistical Administration of the USSR, Gosbank, and in automated management systems of enterprises and agricultural associations."

V. Sereyka, assistant general director of the association on problems of disseminating computer technology:

"Let me add that we not only deliver our machines and computer complexes, but we assist in putting them into operation. The workers of the association start the computers, provide guaranteed servicing and repair. And it must be said that our machines are convenient to operate. There are no special requirements for accommodations or skill of service personnel."

Over the 25 years of existence of the enterprise, without interrupting their work, 343 workers have been graduated from universities, 470 have completed special secondary educational institutions, and 670 workers have had some secondary education.

6610
CSO: 1863/9

FRUNZE INSTITUTE EXPANDS COMPUTER CAPABILITIES

Frunze SOVETSKAYA KIRGIZIYA in Russian 13 Oct 82 p 3

[Excerpts from article by V. Zhuravlev, rector of Frunze Polytechnical Institute, and A. Akadev, chairman of computer department: "Students and Computers"]

[Excerpts] In all faculties of our institute, students are being taught programming and computer solution of engineering problems. In addition, we are training specialists in areas such as design of computer facilities, computer operation and computer center programming.

During this academic year, our student computer center will be made operational. The operation of this facility is organized on the "open door" principle. Students can come in and do their work throughout the school day. The YeS-1020 computer that is the basis for the center was obtained from our patron institution in Leningrad.

The institute has some small computers of the Nairi-3 class. These are used by departmental laboratories.

Leading higher educational institutions of the nation are making extensive use of automated teaching systems.

During this academic year, such a system is being introduced in our institute. One of the institutes of the USSR Academy of Sciences has turned over to us a highly productive M-4030 control computer.

We have discussed what is being done at our institute on computer introduction. Will this contradict the trend of development of the republic interacademic computing center? Definitely not. We will not be duplicating this effort in any way. By the most modest estimates, the institute will need 6,000 hours of machine time per year for teaching and research. Last year we managed to get only 1,219 hours in the republic interacademic computing center.

6610
CSO: 1863/9

COMPUTING NETWORK AND REAL TIME SYSTEM SOFTWARE

Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 4, Jul-Aug 82 pp 126-127

[Item by Boris Vladimirovich Arkhangel'skiy, candidate of physical and mathematical sciences, IK AN USSR [Institute of Cybernetics, Ukrainian SSR Academy of Sciences], Kiev, and Andrey Ivanovich Nikitin, doctor of technical sciences, IK AN USSR, Kiev, in section "Symposiums, Conferences, Meetings"]

[Text] The all-Union "Computer Network and Real Time System Software" conference organized by the USSR State Committee on Science and Technology Central Administration for Computer Technology and Control Systems, the academies of sciences of the USSR and Ukrainian SSR, and the Ukrainian SSR Academy of Sciences Institute of Cybernetics was held from 15 through 17 December 1981 in Kiev.

More than 150 representatives of scientific research institutes, industrial enterprises and VUZ's (a total of 53 organizations) from 23 cities in the country took part in the proceedings of the conference.

Urgent problems were discussed at the conference, relating to the creation of computer network and real time system software (PO) and methods of developing it and using it efficiently, and also practical know-how gained in this field was generalized and recommendations were developed for developers.

Papers presented at the conference were discussed in the following sections: general software of computer networks and real time systems; improving the quality and reliability of software, optimizing programs, and the technical and economic efficiency of software; standardization of software and hierarchy of protocols; and software for organizing the computing process (checking, calculation, dispatching, plannning).

The form for conducting the conference--plenary addresses, bench reports and discussions--was very fruitful in the opinion of the participants.

Seven plenary papers were listened to.

In a paper by V.M. Glushkov and A.I. Nikitin, "Some Problems Relating to Creation and Development of Computer Networks," classes of computer networks were distinguished and problems originating in implementing them were discussed. The "problem of the century" in the opinion of the authors is the creation of a hierarchy of protocols controlling data transfer and the computing and data processing process

itself in the network. The prospects were discussed for the creation of complexes of computers distributed over a limited area, according to the principle of open, so-called local, computer networks. Attention was paid in the paper to the problem of optimum synthesis of the topology of communications subsystems in computer networks and of controlling data streams in these subsystems.

A paper by B.B. Timofev and A.I. Sbitnev, "Principles of Structural Organization and Design of ASUTP [Automated System for Controlling a Technological Process] Software for Complex Processes," was devoted to a discussion of the approach to designing and debugging ASUTP software used at the Kiev Institute of Automation. Designing is performed in layers using a limited set of permissible structures. Designing is performed both from top to bottom (by a mockup redefinition of undeveloped elements) and also from bottom to top (using test interfaces). Questions relating to ensuring the reliability of software created were discussed in detail and key problems to be solved in the design process and the basic results of introduction were described.

In a paper by V.V. Lipayev, "Software Quality Indicators," the key concepts of criteria for the quality of programs and factors influencing their values were discussed. Structural quality indicators were distinguished, whose content depends but slightly on the functional purpose of programs. Characteristics of various indicators of the complexity of programs in the design process and in use were presented and key concepts relating to the correctness of programs and their relationship to types of standards and methods of establishing correctness were formulated. Application of the key concepts of the theory of reliability was refined for complexes of programs.

In a paper by V.N. Krinitskiy, N.A. Krinitskiy, G.A. Mironov and G.D. Frolov, "Structure of Computer Network Software," a model of a computer network is discussed. This model is adaptive and requires from the user minimum knowledge of the information structure, which is open both for bringing in new computers and for expanding the class of problems which can be solved automatically (standard problems). The authors made an analysis and classification of functional structures and discussed data bank structures known at the present time. The possibilities of integrating information systems were also analyzed, unification of which into a multileveled ensemble can be regarded as a model of a computer network.

A paper by A.I. Il'yushin, A.I. Myamlin and Vs.S. Shtarkman, "Software Upper Level Design Principles," was devoted to one of the most urgent problems in the development of modern networks. The authors developed a fundamentally new system for designing software upper levels, based on ideas relating to interaction of abstract entities (subsystems, clusters and packages of procedures). The basic mechanism of this interaction is initiation of any procedure in a remote abstract entity (subsystem) with the transfer as arguments and results of both built-in entities and abstract ones.

In a paper by L.A. Serebrovskiy, V.V. Lipayev and M.A. Minayev, "Unification of Languages for Designing Complexes of Programs for Special-Purpose Computers," five groups of interrelated languages for the design of software are singled out, oriented to specific stages of software development. It is suggested that the synthesis of these languages be built on the basis of a unified high-level

programming language making possible a number of properties determined by the features of programming for special-purpose computers.

In a paper by B.V. Arkhangel'skiy, "Practical Possibilities for Optimizing Programs," was formulated the understanding of the problem, arrived at at the present time, of optimizing programs and of basic approaches to solving it. The author discussed the optimization criterion, application strategies, the optimization result and requirements imposed on optimization procedures, and the relationship between optimization and other stages in the creation of programs. Levels of languages and areas of a program within whose limits optimization transformations are performed were discussed.

Also discussed at the conference were 91 principal bench reports (the theses of these reports have been published) and 27 supplementary bench reports.

In the discussion there was an exchange of opinions on problems and further ways of developing software for computing networks and real time systems. The heightened level of the theoretical and experimental studies presented at the conference was emphasized, as well as their fundamental nature and practical significance, and extensive introduction in industry.

The problems discussed in the plenary and bench reports and in the discussion were reflected in the resolution made by the conference, in which it was stated in particular that scientific associates and engineering and technical personnel of enterprises, scientific research institutes, design bureaus and academic, planning and educational institutes had become acquainted with the status of problems relating to creation and development of computer network and real time system software and had exchanged working know-how. A decision was made to organize the next conference in 1983.

The participants in the conference believe that the exchange of opinions and results relating to the main problems discussed was very thorough and helpful and noted the high scientific organizational level with which the conference was conducted.

8831
CSO: 1863/1

END